CRISP-DM PROYECTO

November 1, 2025

1 Unidad 1 – Fases 1 a 3 (CRISP-DM)

1.0.1 Proyecto: Precios componentes computadora 2025

-Business Understanding → Data Understanding → Data Preparation - **Integrantes:** [Esaú Morales , Roxwell Ramos , Joshua Alfonzo]

1.1 Fase 1. Business Understanding

1.1.1 1. Objective

Construir un modelo de **regresión supervisada** que prediga el **precio de una computadora** (`price`) en función de sus características técnicas (**RAM, almacenamiento, procesador, GPU, pantalla, marca, tipo, etc.**).
El objetivo de negocio es **estimar precios de mercado con precisión**, optimizando estrategias de venta, compras o recomendaciones de productos.

Success Criteria (SC)

- **De negocio:** permitir una estimación confiable del precio de computadoras nuevas o usadas para mejorar decisiones de pricing o recomendación.
- **De Machine Learning (en conjunto de test):**
  - R^2 **0.85** (explicación de la varianza del precio).
  - **RMSE** **500** (error medio cuadrático raíz aceptable).
  - **MAE** **300** (error absoluto medio).
  - Mostrar **importancia de variables** (features más influyentes en el precio).

1.2 Fase 2. Data Understanding

1.2.1 2. Data collection

[1]:
```
# ========================================================
#  Data Collection
# ========================================================
# En esta sección se realiza la carga del dataset "All Computer Prices (2025)",
```

```
# el cual contiene 100,000 registros con información técnica de equipos de␣
 ↪cómputo
# (RAM, CPU, GPU, almacenamiento, pantalla, marca, etc.) y su precio.
# El objetivo es preparar la base de datos para su posterior análisis y␣
 ↪modelado.

# Importar librerías
import pandas as pd
import numpy as np

# Cargar dataset
df = pd.read_csv("computer_prices_all.csv",
                 encoding='utf-8',    # codificación estándar
                 sep=',',             # separador CSV
                 header=0,            # primera fila = nombres de columnas
                 index_col=None)      # no se usa índice personalizado

# Dimensiones del dataset
print("Número de filas y columnas:", df.shape)

# Mostrar primeras filas
df.head()
```

```
Número de filas y columnas: (100000, 33)
```

[1]:

	device_type	brand	model	release_year	os	form_factor	…	resolution	refresh_hz	battery_wh	charger_watts	psu_watts	wifi	bluetooth	weight_kg	warranty_months	price
0	Desktop	Samsung	Samsung Forge XDI	2022	Windows	ATX	…	2560x1440	90	0	0	750	Wi-Fi 6	5.1			
1	Laptop	Samsung	Samsung Pro KM8	2022	Windows	Mainstream	…	1920x1080	90	56	120	0	Wi-Fi 6	5.3			
2	Desktop	Lenovo	Lenovo Strix BIE	2024	macOS	SFF	…	3440x1440	120	0	0	850	Wi-Fi 6	5.0			
3	Desktop	Dell	Dell Cube AXR	2024	Windows	ATX	…	3440x1440	120	0	0	650	Wi-Fi 6	5.2			
4	Laptop	Gigabyte	Gigabyte Pro IX1	2024	Linux	Gaming	…	2560x1600	90	80	90	0	Wi-Fi 6	5.2			

	cpu_brand	cpu_model	cpu_tier	cpu_cores
0	Intel	Intel i5-11129	3	12
1	Intel	Intel i7-11114	4	12
2	AMD	AMD Ryzen 5 5168	2	8
3	AMD	AMD Ryzen 5 7550	2	6
4	AMD	AMD Ryzen 7 6230	5	16

```
0      11.00              36  1383.99
1       2.03              12  2274.99
2       7.00              24  1879.99
3       6.00              36  1331.99
4       1.50              12  2681.99

[5 rows x 33 columns]
```

[2]:
```python
# ============================================================
#   Resumen general del dataset de precios de computadoras
# ============================================================

# Información general del DataFrame (tipos de datos, nulos, uso de memoria)
df.info()

# Nombres de las columnas
print('Columnas del dataset:', df.columns.tolist())

# Dimensiones del dataset (filas, columnas)
print('Dimensiones del dataset:', df.shape)

# Mostrar las primeras 10 filas como muestra
df.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 33 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   device_type          100000 non-null  object
 1   brand                100000 non-null  object
 2   model                100000 non-null  object
 3   release_year         100000 non-null  int64
 4   os                   100000 non-null  object
 5   form_factor          100000 non-null  object
 6   cpu_brand            100000 non-null  object
 7   cpu_model            100000 non-null  object
 8   cpu_tier             100000 non-null  int64
 9   cpu_cores            100000 non-null  int64
 10  cpu_threads          100000 non-null  int64
 11  cpu_base_ghz         100000 non-null  float64
 12  cpu_boost_ghz        100000 non-null  float64
 13  gpu_brand            100000 non-null  object
 14  gpu_model            100000 non-null  object
 15  gpu_tier             100000 non-null  int64
 16  vram_gb              100000 non-null  int64
 17  ram_gb               100000 non-null  int64
 18  storage_type         100000 non-null  object
```

```
 19  storage_gb           100000 non-null  int64
 20  storage_drive_count  100000 non-null  int64
 21  display_type         100000 non-null  object
 22  display_size_in      100000 non-null  float64
 23  resolution           100000 non-null  object
 24  refresh_hz           100000 non-null  int64
 25  battery_wh           100000 non-null  int64
 26  charger_watts        100000 non-null  int64
 27  psu_watts            100000 non-null  int64
 28  wifi                 100000 non-null  object
 29  bluetooth            100000 non-null  float64
 30  weight_kg            100000 non-null  float64
 31  warranty_months      100000 non-null  int64
 32  price                100000 non-null  float64
dtypes: float64(6), int64(14), object(13)
memory usage: 25.2+ MB
Columnas del dataset: ['device_type', 'brand', 'model', 'release_year', 'os',
'form_factor', 'cpu_brand', 'cpu_model', 'cpu_tier', 'cpu_cores', 'cpu_threads',
'cpu_base_ghz', 'cpu_boost_ghz', 'gpu_brand', 'gpu_model', 'gpu_tier',
'vram_gb', 'ram_gb', 'storage_type', 'storage_gb', 'storage_drive_count',
'display_type', 'display_size_in', 'resolution', 'refresh_hz', 'battery_wh',
'charger_watts', 'psu_watts', 'wifi', 'bluetooth', 'weight_kg',
'warranty_months', 'price']
Dimensiones del dataset: (100000, 33)
```

[2]:

	device_type	brand	model	release_year	os	form_factor
0	Desktop	Samsung	Samsung Forge XDI	2022	Windows	ATX
1	Laptop	Samsung	Samsung Pro KM8	2022	Windows	Mainstream
2	Desktop	Lenovo	Lenovo Strix BIE	2024	macOS	SFF
3	Desktop	Dell	Dell Cube AXR	2024	Windows	ATX
4	Laptop	Gigabyte	Gigabyte Pro IX1	2024	Linux	Gaming
5	Desktop	MSI	MSI Think KSG	2025	Windows	ATX
6	Desktop	Apple	Apple Arena R5Q	2024	Windows	ATX
7	Desktop	Apple	Apple Station EWP	2023	Windows	Micro-ATX
8	Laptop	Dell	Dell Creator GIQ	2024	Windows	Mainstream
9	Laptop	Lenovo	Lenovo Blade MIZ	2025	Windows	Ultrabook

	cpu_brand	cpu_model	cpu_tier	cpu_cores	…	resolution
0	Intel	Intel i5-11129	3	12	…	2560x1440
1	Intel	Intel i7-11114	4	12	…	1920x1080
2	AMD	AMD Ryzen 5 5168	2	8	…	3440x1440
3	AMD	AMD Ryzen 5 7550	2	6	…	3440x1440
4	AMD	AMD Ryzen 7 6230	5	16	…	2560x1600
5	Intel	Intel i7-10369	5	16	…	2560x1440
6	Apple	Apple M2	2	6	…	2560x1440
7	Apple	Apple M2 Pro	3	8	…	2560x1440
8	Intel	Intel i9-14473	6	26	…	2560x1600

```
9       AMD  AMD Ryzen 3 4374         1          4  …   3840x2160

   refresh_hz  battery_wh charger_watts psu_watts     wifi  bluetooth  \
0          90           0             0       750  Wi-Fi 6        5.1
1          90          56           120         0  Wi-Fi 6        5.3
2         120           0             0       850  Wi-Fi 6        5.0
3         120           0             0       650  Wi-Fi 6        5.2
4          90          80            90         0  Wi-Fi 6        5.2
5          90           0             0      1000  Wi-Fi 5        5.0
6          60           0             0       850  Wi-Fi 6        5.1
7          60           0             0       650  Wi-Fi 6        5.0
8          60          80           240         0  Wi-Fi 5        5.0
9         120          60            45         0  Wi-Fi 6        5.3

   weight_kg warranty_months    price
0      11.00              36  1383.99
1       2.03              12  2274.99
2       7.00              24  1879.99
3       6.00              36  1331.99
4       1.50              12  2681.99
5       9.00              36  2751.99
6       9.00              24  1609.99
7       8.00              12  2139.99
8       1.17              48  2953.99
9       1.50              24  1653.99

[10 rows x 33 columns]
```

1.2.2 3. Descriptive analysis

3.1 Análisis descriptivo

[3]:
```
# ============================================================
#   3. Descriptive Analysis
# ============================================================
# Resumen estadístico general de todas las variables
# (numéricas y categóricas) para obtener una visión inicial
# del comportamiento de los datos.

df.describe(include="all").T
```

[3]:
```
                       count unique              top   freq        mean  \
device_type           100000      2           Laptop  59844         NaN
brand                 100000     10           Lenovo  15992         NaN
model                 100000  99036  HP Creator R41      3         NaN
release_year         100000.0    NaN              NaN    NaN  2022.32085
os                    100000      4          Windows  71817         NaN
form_factor           100000     10       Mainstream  17819         NaN
```

cpu_brand	100000	3	Intel	52774	NaN	
cpu_model	100000	26971	Apple M2 Pro	1389	NaN	
cpu_tier	100000.0	NaN	NaN	NaN	3.15349	
cpu_cores	100000.0	NaN	NaN	NaN	10.51574	
cpu_threads	100000.0	NaN	NaN	NaN	19.3727	
cpu_base_ghz	100000.0	NaN	NaN	NaN	2.591322	
cpu_boost_ghz	100000.0	NaN	NaN	NaN	3.53131	
gpu_brand	100000	4	NVIDIA	54712	NaN	
gpu_model	100000	49	Apple Integrated	18922	NaN	
gpu_tier	100000.0	NaN	NaN	NaN	2.99135	
vram_gb	100000.0	NaN	NaN	NaN	6.15218	
ram_gb	100000.0	NaN	NaN	NaN	39.7064	
storage_type	100000	4	NVMe	45059	NaN	
storage_gb	100000.0	NaN	NaN	NaN	903.936	
storage_drive_count	100000.0	NaN	NaN	NaN	1.52498	
display_type	100000	6	LED	32000	NaN	
display_size_in	100000.0	NaN	NaN	NaN	20.126655	
resolution	100000	6	1920x1080	47993	NaN	
refresh_hz	100000.0	NaN	NaN	NaN	98.46486	
battery_wh	100000.0	NaN	NaN	NaN	41.81347	
charger_watts	100000.0	NaN	NaN	NaN	61.38345	
psu_watts	100000.0	NaN	NaN	NaN	272.5205	
wifi	100000	4	Wi-Fi 6	46149	NaN	
bluetooth	100000.0	NaN	NaN	NaN	5.084764	
weight_kg	100000.0	NaN	NaN	NaN	4.289699	
warranty_months	100000.0	NaN	NaN	NaN	22.20036	
price	100000.0	NaN	NaN	NaN	1928.76422	

	std	min	25%	50%	75%	max
device_type	NaN	NaN	NaN	NaN	NaN	NaN
brand	NaN	NaN	NaN	NaN	NaN	NaN
model	NaN	NaN	NaN	NaN	NaN	NaN
release_year	2.025761	2018.0	2021.0	2023.0	2024.0	2025.0
os	NaN	NaN	NaN	NaN	NaN	NaN
form_factor	NaN	NaN	NaN	NaN	NaN	NaN
cpu_brand	NaN	NaN	NaN	NaN	NaN	NaN
cpu_model	NaN	NaN	NaN	NaN	NaN	NaN
cpu_tier	1.373175	1.0	2.0	3.0	4.0	6.0
cpu_cores	5.044092	4.0	6.0	8.0	14.0	28.0
cpu_threads	9.718426	4.0	12.0	16.0	24.0	56.0
cpu_base_ghz	0.336435	2.0	2.4	2.6	2.8	3.4
cpu_boost_ghz	0.350024	2.8	3.3	3.5	3.8	4.5
gpu_brand	NaN	NaN	NaN	NaN	NaN	NaN
gpu_model	NaN	NaN	NaN	NaN	NaN	NaN
gpu_tier	1.459643	1.0	2.0	3.0	4.0	6.0
vram_gb	3.964926	0.0	4.0	6.0	8.0	16.0
ram_gb	31.902684	8.0	16.0	32.0	64.0	144.0

storage_type	NaN	NaN	NaN	NaN	NaN	NaN
storage_gb	774.243654	256.0	512.0	512.0	1024.0	4096.0
storage_drive_count	0.797284	1.0	1.0	1.0	2.0	4.0
display_type	NaN	NaN	NaN	NaN	NaN	NaN
display_size_in	6.709577	13.3	14.0	16.0	27.0	34.0
resolution	NaN	NaN	NaN	NaN	NaN	NaN
refresh_hz	43.301652	60.0	60.0	90.0	120.0	240.0
battery_wh	35.868841	0.0	0.0	56.0	70.0	99.0
charger_watts	62.795034	0.0	0.0	65.0	90.0	240.0
psu_watts	354.686355	0.0	0.0	0.0	650.0	1200.0
wifi	NaN	NaN	NaN	NaN	NaN	NaN
bluetooth	0.245977	4.2	5.0	5.1	5.2	5.3
weight_kg	3.814628	0.92	1.5	2.0	7.0	16.0
warranty_months	10.2319	12.0	12.0	24.0	24.0	48.0
price	580.492689	372.99	1503.99	1863.99	2287.99	10984.99

[4]:
```
# ============================================================
#  4. Separación de datos por tipo de variable
# ============================================================
# Separamos las columnas numéricas y categóricas
# (si existieran columnas de fecha, podrían tratarse como categóricas también).

numeric_data = df.select_dtypes(include=[np.number])    # columnas numéricas
categor_data = df.select_dtypes(exclude=[np.number])    # columnas no numéricas

print("Hay {} columnas numéricas y {} columnas categóricas en el dataset."
      .format(numeric_data.shape[1], categor_data.shape[1]))

# Mostrar ejemplos
print("\nColumnas numéricas:", numeric_data.columns.tolist())
print("\nColumnas categóricas:", categor_data.columns.tolist())
```

```
Hay 20 columnas numéricas y 13 columnas categóricas en el dataset.

Columnas numéricas: ['release_year', 'cpu_tier', 'cpu_cores', 'cpu_threads',
'cpu_base_ghz', 'cpu_boost_ghz', 'gpu_tier', 'vram_gb', 'ram_gb', 'storage_gb',
'storage_drive_count', 'display_size_in', 'refresh_hz', 'battery_wh',
'charger_watts', 'psu_watts', 'bluetooth', 'weight_kg', 'warranty_months',
'price']

Columnas categóricas: ['device_type', 'brand', 'model', 'os', 'form_factor',
'cpu_brand', 'cpu_model', 'gpu_brand', 'gpu_model', 'storage_type',
'display_type', 'resolution', 'wifi']
```

[5]:
```
# ============================================================
#  5. Análisis descriptivo de variables numéricas
# ============================================================
# Se obtiene un resumen estadístico solo de las variables numéricas
```

```
# (promedio, desviación estándar, valores mínimos, máximos, cuartiles, etc.)

numeric_data.describe().T
```

[5]:

	count	mean	std	min	25%	50%	75%	max
release_year	100000.0	2022.320850	2.025761	2018.00	2021.00	2023.00	2024.00	2025.00
cpu_tier	100000.0	3.153490	1.373175	1.00	2.00	3.00	4.00	6.00
cpu_cores	100000.0	10.515740	5.044092	4.00	6.00	8.00	14.00	28.00
cpu_threads	100000.0	19.372700	9.718426	4.00	12.00	16.00	24.00	56.00
cpu_base_ghz	100000.0	2.591322	0.336435	2.00	2.40	2.60	2.80	3.40
cpu_boost_ghz	100000.0	3.531310	0.350024	2.80	3.30	3.50	3.80	4.50
gpu_tier	100000.0	2.991350	1.459643	1.00	2.00	3.00	4.00	6.00
vram_gb	100000.0	6.152180	3.964926	0.00	4.00	6.00	8.00	16.00
ram_gb	100000.0	39.706400	31.902684	8.00	16.00	32.00	64.00	144.00
storage_gb	100000.0	903.936000	774.243654	256.00	512.00	512.00	1024.00	4096.00
storage_drive_count	100000.0	1.524980	0.797284	1.00	1.00	1.00	2.00	4.00
display_size_in	100000.0	20.126655	6.709577	13.30	14.00	16.00	27.00	34.00
refresh_hz	100000.0	98.464860	43.301652	60.00	60.00	90.00	120.00	240.00
battery_wh	100000.0	41.813470	35.868841	0.00	0.00	56.00	70.00	99.00
charger_watts	100000.0	61.383450	62.795034	0.00	0.00	65.00	90.00	240.00
psu_watts	100000.0	272.520500	354.686355	0.00	0.00	0.00	650.00	1200.00
bluetooth	100000.0	5.084764	0.245977	4.20	5.00	5.10	5.20	5.30
weight_kg	100000.0	4.289699	3.814628	0.92	1.50	2.00	7.00	16.00
warranty_months	100000.0	22.200360	10.231900	12.00	12.00	24.00	24.00	48.00
price	100000.0	1928.764220	580.492689	372.99	1503.99	1863.99	2287.99	10984.99

[6]:
```
# ============================================================
#  6. Análisis descriptivo de variables categóricas
# ============================================================
# Se obtiene un resumen estadístico de las variables categóricas,
# mostrando la cantidad de valores únicos, el más frecuente (top)
# y su frecuencia (freq).

categor_data.describe(include="all").T
```

[6]:

	count	unique	top	freq
device_type	100000	2	Laptop	59844
brand	100000	10	Lenovo	15992
model	100000	99036	HP Creator R41	3
os	100000	4	Windows	71817
form_factor	100000	10	Mainstream	17819
cpu_brand	100000	3	Intel	52774
cpu_model	100000	26971	Apple M2 Pro	1389
gpu_brand	100000	4	NVIDIA	54712
gpu_model	100000	49	Apple Integrated	18922
storage_type	100000	4	NVMe	45059
display_type	100000	6	LED	32000
resolution	100000	6	1920x1080	47993
wifi	100000	4	Wi-Fi 6	46149

[7]:
```
# ============================================================
#  7. Conteo de valores únicos en variables categóricas
# ============================================================
# Se realiza un conteo de frecuencia para las principales variables categóricas
# con el fin de identificar su distribución y posibles desequilibrios.

#  Ajusta los nombres según las columnas categóricas de tu dataset
for col in ['brand', 'processor', 'gpu', 'storage_type']:
    if col in df.columns:
        print(f"\n{col}:\n", df[col].value_counts())

#  Distribución de la variable objetivo (target)
# En este caso, el objetivo es el precio (variable continua), por lo tanto
# solo se muestra su resumen estadístico.
print("\n Resumen estadístico de la variable objetivo (price):\n")
print(df['price'].describe())
```

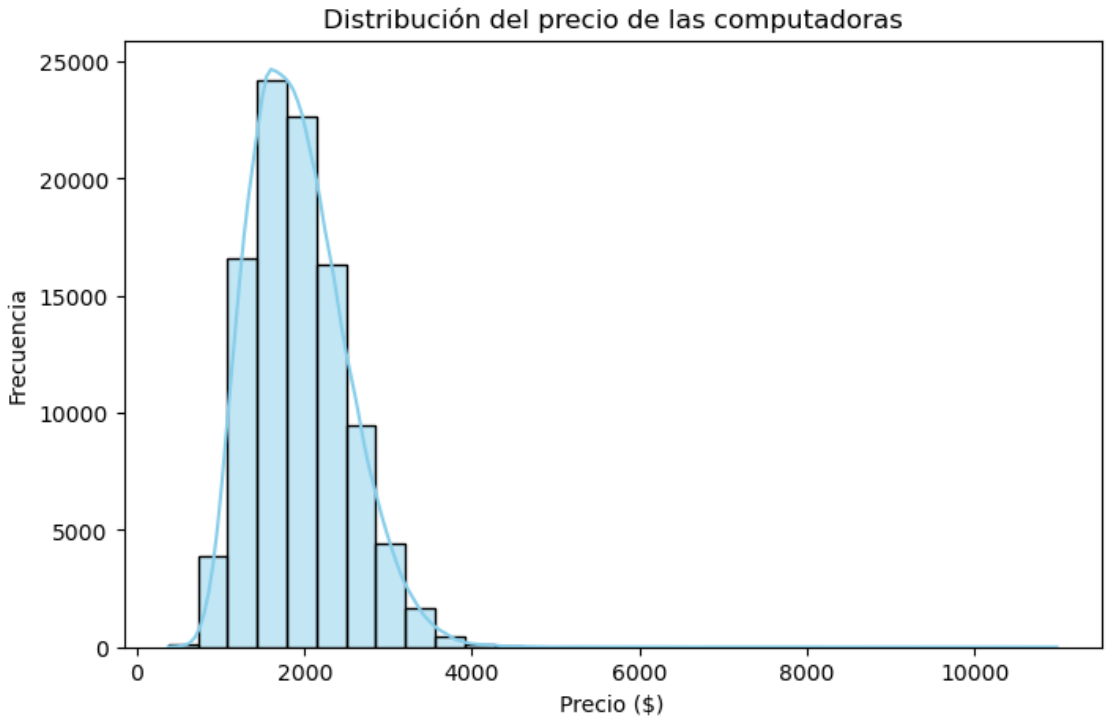
```
brand:
 brand
Lenovo      15992
HP          14114
Dell        14005
Apple       11915
```

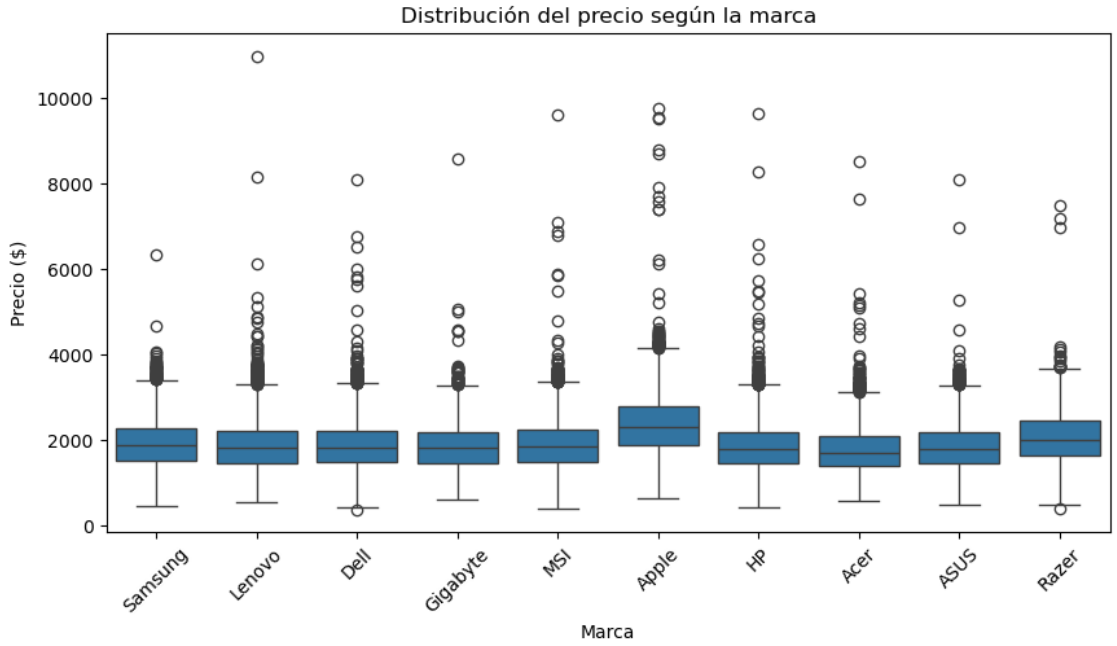
```
ASUS         10159
Acer          9925
Samsung       8066
MSI           7891
Gigabyte      4900
Razer         3033
Name: count, dtype: int64

storage_type:
 storage_type
NVMe       45059
SSD        24937
HDD        15023
Hybrid     14981
Name: count, dtype: int64

 Resumen estadístico de la variable objetivo (price):

count    100000.000000
mean       1928.764220
std         580.492689
min         372.990000
25%        1503.990000
50%        1863.990000
75%        2287.990000
max       10984.990000
Name: price, dtype: float64
```

3.2 Análisis descriptivo (gráficos)

[8]:
```python
# ============================================================
#  8. Visualización inicial de variables
# ============================================================
import matplotlib.pyplot as plt
import seaborn as sns

#  Histograma del precio
plt.figure(figsize=(8,5))
sns.histplot(df['price'], bins=30, kde=True, color='skyblue')
plt.title("Distribución del precio de las computadoras")
plt.xlabel("Precio ($)")
plt.ylabel("Frecuencia")
plt.show()

#  Boxplot de precio por marca
plt.figure(figsize=(10,5))
if 'brand' in df.columns:
    sns.boxplot(x='brand', y='price', data=df)
```

```
    plt.title("Distribución del precio según la marca")
    plt.xlabel("Marca")
    plt.ylabel("Precio ($)")
    plt.xticks(rotation=45)
    plt.show()

#  Boxplot de precio por tipo de procesador
plt.figure(figsize=(10,5))
if 'processor' in df.columns:
    sns.boxplot(x='processor', y='price', data=df)
    plt.title("Distribución del precio según el procesador")
    plt.xlabel("Procesador")
    plt.ylabel("Precio ($)")
    plt.xticks(rotation=45)
    plt.show()

#  Conteo de tarjetas gráficas (GPU)
plt.figure(figsize=(10,5))
if 'gpu' in df.columns:
    sns.countplot(data=df, x='gpu', order=df['gpu'].value_counts().index)
    plt.title("Distribución de tipos de GPU")
    plt.xlabel("GPU")
    plt.ylabel("Cantidad de equipos")
    plt.xticks(rotation=45)
    plt.show()
```





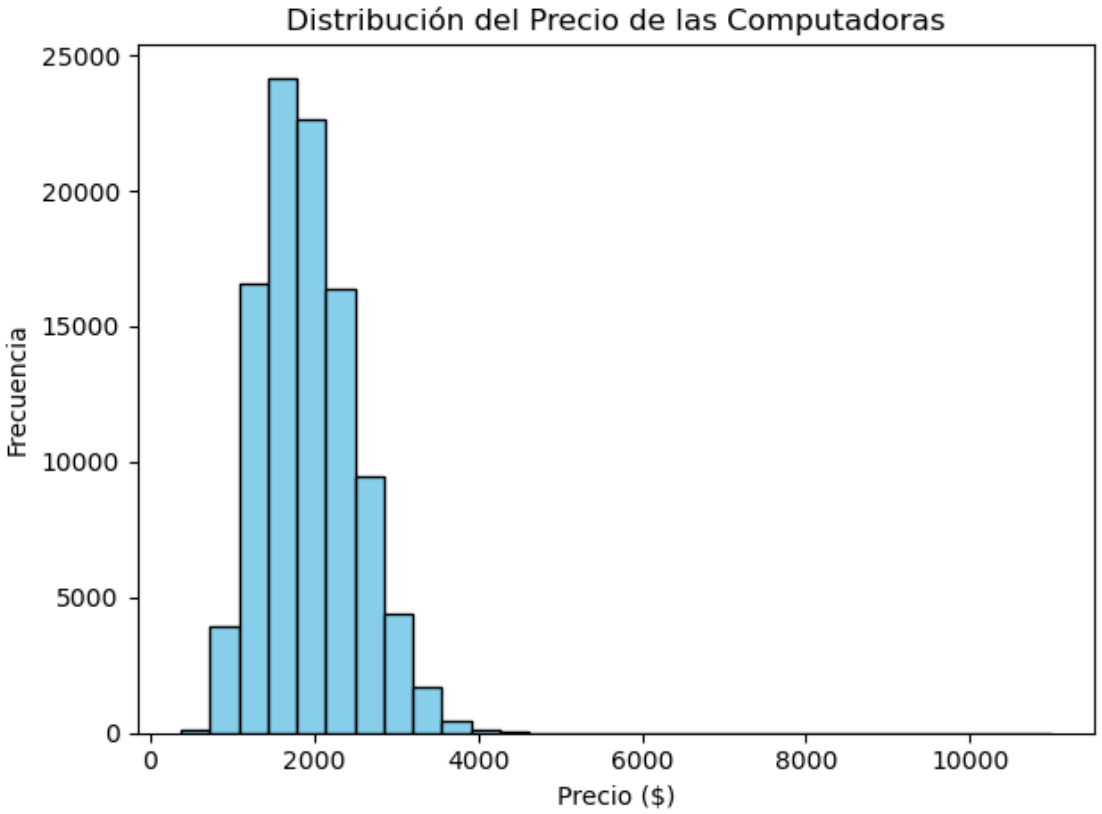
<Figure size 1000x500 with 0 Axes>

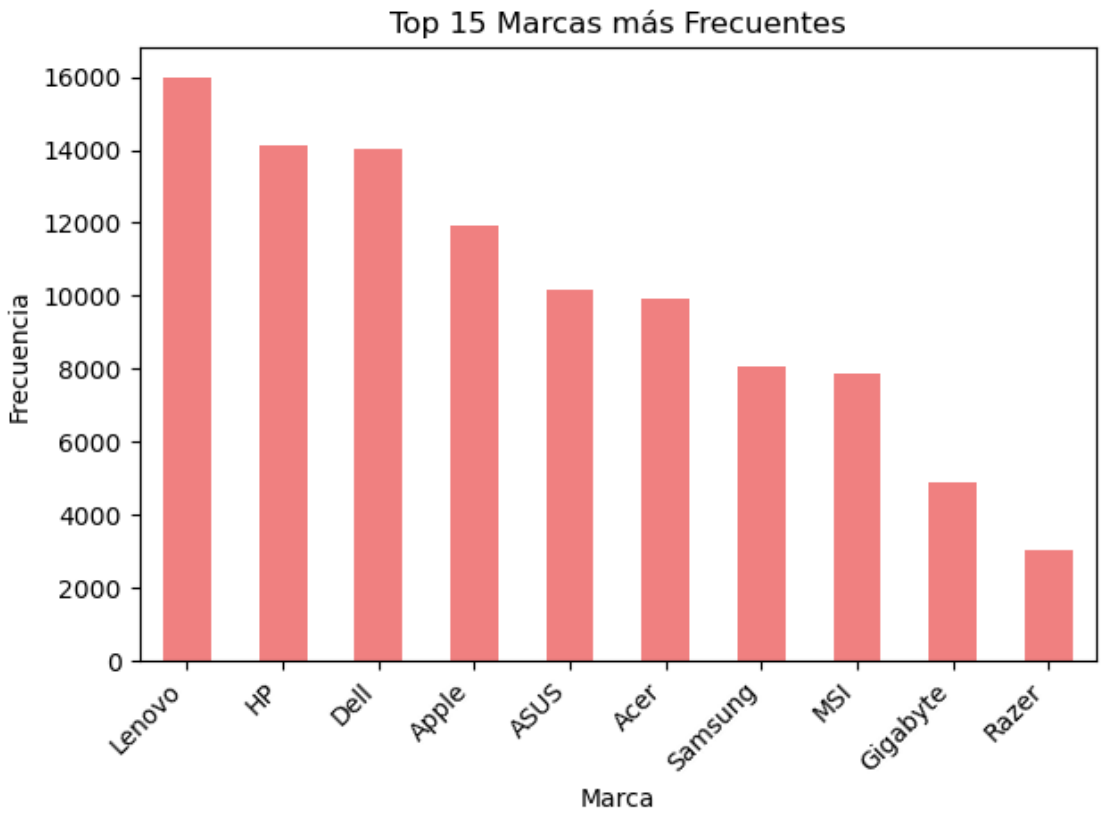
<Figure size 1000x500 with 0 Axes>

[9]:
```
# ============================================================
#   9. Visualización opcional (recomendada)
# ============================================================
# Se visualizan distribuciones y frecuencias de variables clave
# para entender mejor los patrones antes del modelado.

import matplotlib.pyplot as plt

# ==========================
# Distribución de la variable objetivo (price)
# ==========================
df['price'].plot(kind='hist', bins=30, edgecolor='black', color='skyblue')
plt.title('Distribución del Precio de las Computadoras')
plt.xlabel('Precio ($)')
plt.ylabel('Frecuencia')
plt.tight_layout()
plt.show()

# ==========================
# Histogramas de variables numéricas
```

```
# ==========================
# RAM
if 'ram' in df.columns:
    df['ram'].plot(kind='hist', bins=30, edgecolor='black', color='orange')
    plt.title('Distribución de la Memoria RAM')
    plt.xlabel('RAM (GB)')
    plt.ylabel('Frecuencia')
    plt.tight_layout()
    plt.show()

# Almacenamiento
if 'storage' in df.columns:
    df['storage'].plot(kind='hist', bins=30, edgecolor='black', color='green')
    plt.title('Distribución del Almacenamiento')
    plt.xlabel('Almacenamiento (GB o TB)')
    plt.ylabel('Frecuencia')
    plt.tight_layout()
    plt.show()

# Tamaño de pantalla
if 'screen_size' in df.columns:
    df['screen_size'].plot(kind='hist', bins=30, edgecolor='black',␣
 ↪color='purple')
    plt.title('Distribución del Tamaño de Pantalla')
    plt.xlabel('Pulgadas')
    plt.ylabel('Frecuencia')
    plt.tight_layout()
    plt.show()

# ==========================
# Gráficos de barras para variables categóricas
# ==========================
# Marca
if 'brand' in df.columns:
    df['brand'].value_counts().head(15).plot(kind='bar', color='lightcoral')
    plt.title('Top 15 Marcas más Frecuentes')
    plt.xlabel('Marca')
    plt.ylabel('Frecuencia')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

# Procesador
if 'processor' in df.columns:
    df['processor'].value_counts().head(15).plot(kind='bar', color='lightblue')
    plt.title('Top 15 Procesadores más Frecuentes')
    plt.xlabel('Procesador')
```

```
    plt.ylabel('Frecuencia')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

# GPU
if 'gpu' in df.columns:
    df['gpu'].value_counts().head(15).plot(kind='bar', color='gold')
    plt.title('Top 15 Tipos de GPU más Frecuentes')
    plt.xlabel('GPU')
    plt.ylabel('Frecuencia')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

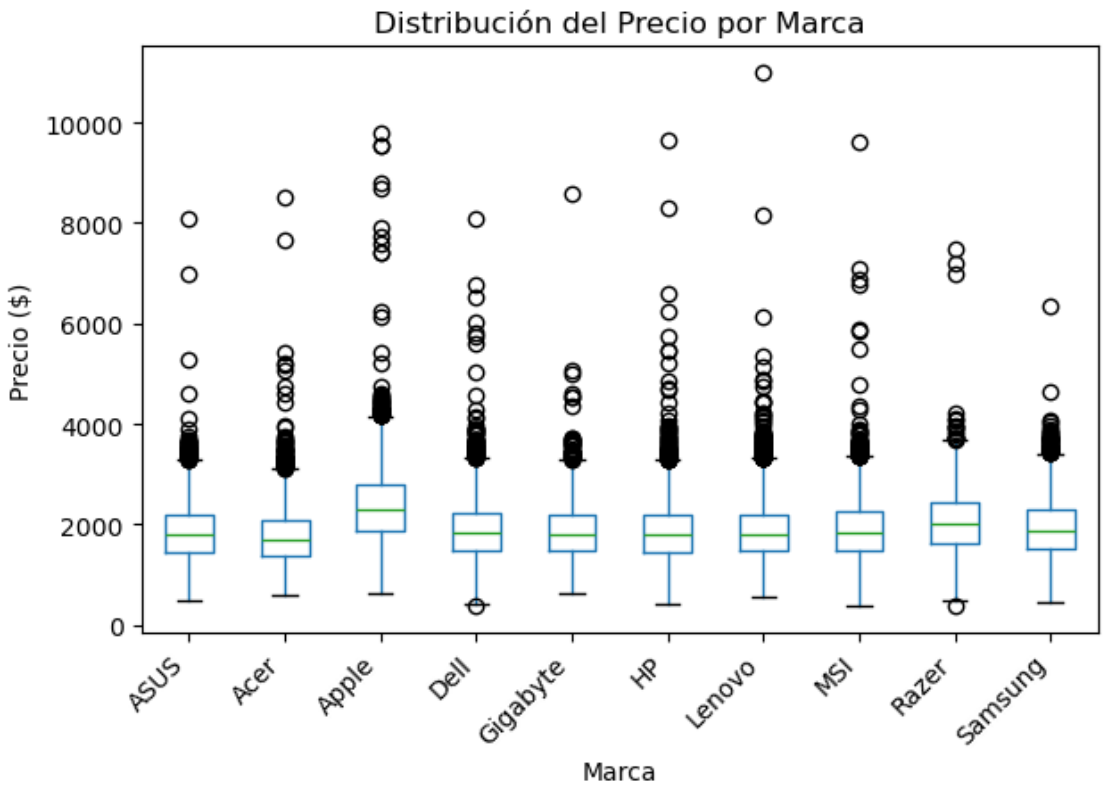




[10]:
```python
# ============================================================
#   10. Boxplots del precio según variables categóricas
# ============================================================
import matplotlib.pyplot as plt

# ========================
# Boxplot de precio por marca
# ========================
if 'brand' in df.columns:
    plt.figure(figsize=(10,5))
    df.boxplot(column='price', by='brand', grid=False)
    plt.title('Distribución del Precio por Marca')
    plt.suptitle('')
    plt.xlabel('Marca')
    plt.ylabel('Precio ($)')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

# ========================
```

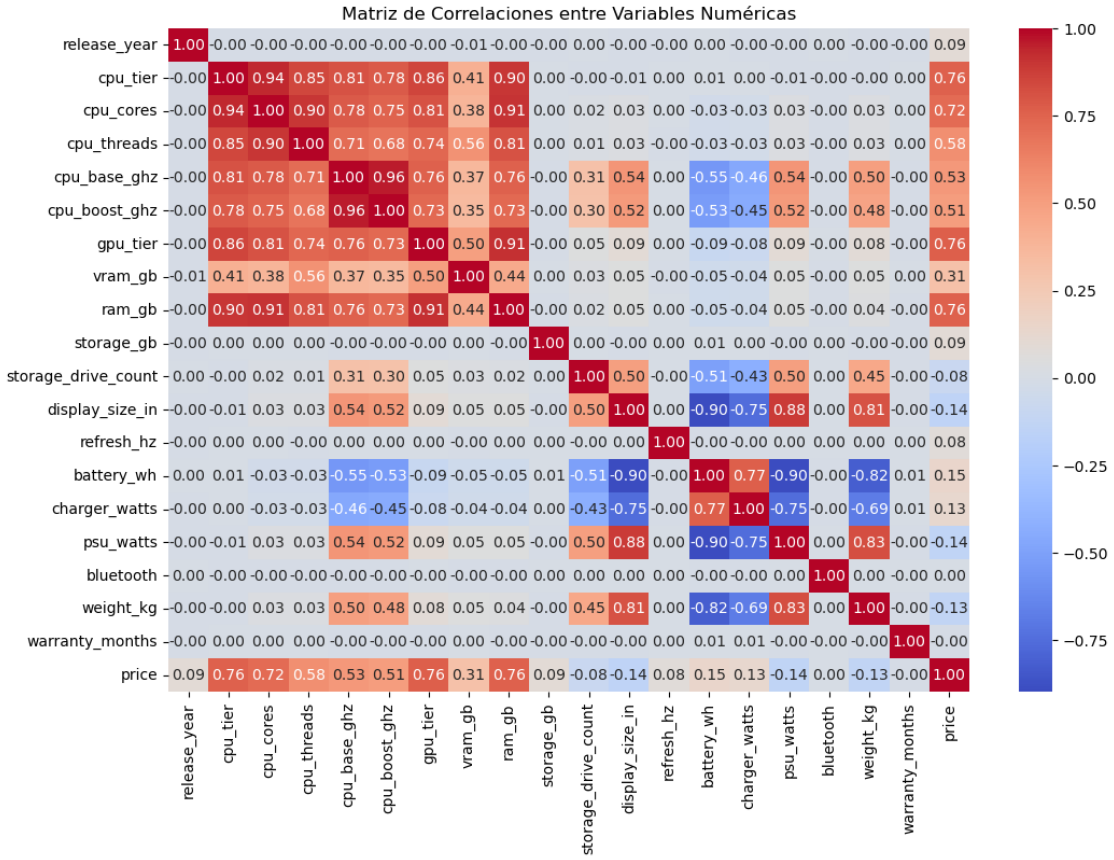
```python
# Boxplot de precio por procesador
# ========================
if 'processor' in df.columns:
    plt.figure(figsize=(10,5))
    df.boxplot(column='price', by='processor', grid=False)
    plt.title('Distribución del Precio por Procesador')
    plt.suptitle('')
    plt.xlabel('Procesador')
    plt.ylabel('Precio ($)')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

# ========================
# Boxplot de precio por GPU
# ========================
if 'gpu' in df.columns:
    plt.figure(figsize=(10,5))
    df.boxplot(column='price', by='gpu', grid=False)
    plt.title('Distribución del Precio por GPU')
    plt.suptitle('')
    plt.xlabel('GPU')
    plt.ylabel('Precio ($)')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```



3.3 Análisis exploratorio (con estadísticas)

[11]:
```
# ============================================================
#   11. Análisis exploratorio: Correlaciones numéricas
# ============================================================
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,8))
corr_matrix = df.corr(numeric_only=True)    # solo variables numéricas
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Matriz de Correlaciones entre Variables Numéricas")
plt.show()
```



1.2.3 Interpretación de la Matriz de Correlaciones

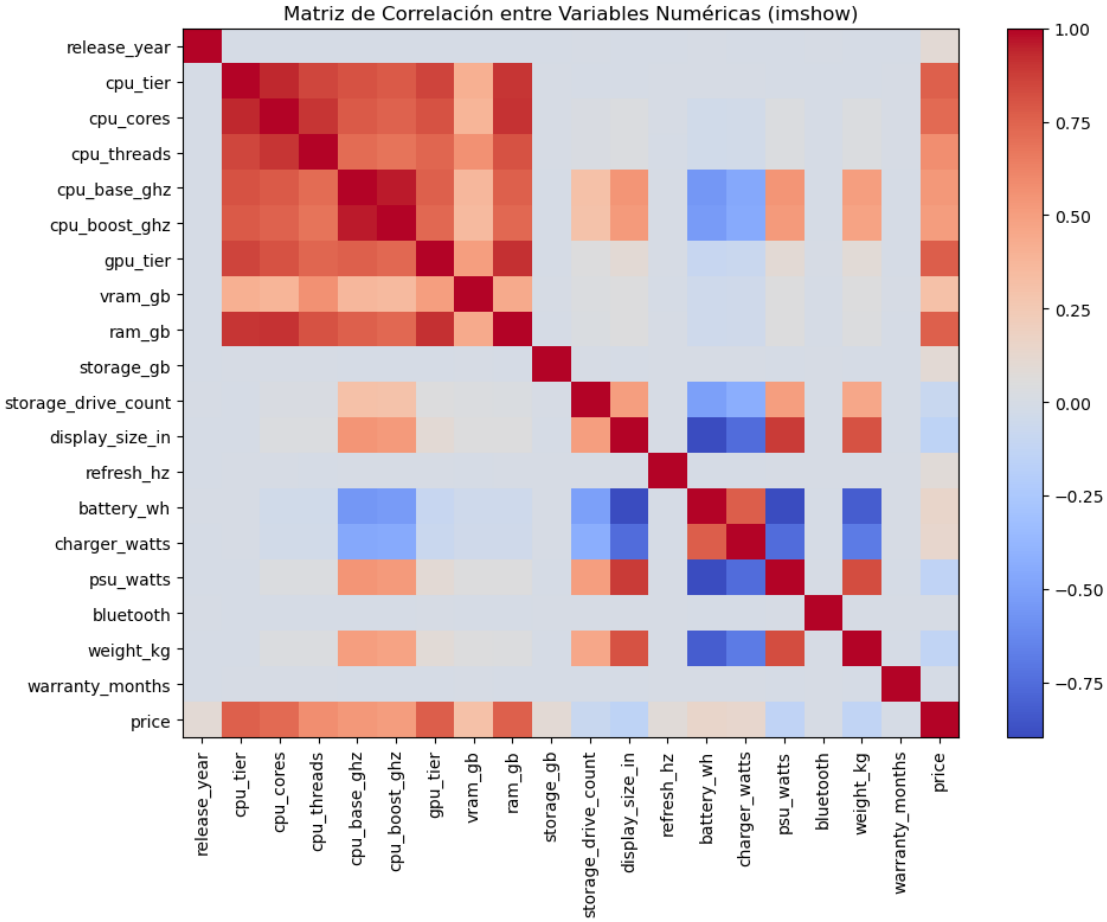
El mapa de calor muestra las correlaciones entre las variables numéricas del dataset de **precios de computadoras**.
A partir de la matriz se observan los siguientes puntos clave:

- **La variable objetivo `price`** presenta una **alta correlación positiva** con:
  - `cpu_tier`, `gpu_tier` y `ram_gb` (0.70–0.76), lo que indica que los equipos con mejor CPU, GPU y mayor RAM tienden a tener precios más altos.
  - `cpu_cores`, `cpu_threads` y `cpu_base_ghz` también muestran correlaciones moderadas (0.50–0.60), reforzando que el rendimiento del procesador influye en el valor final.
- **Variables como `battery_wh`, `charger_watts` y `psu_watts`** muestran **correlaciones negativas o muy bajas**, lo que sugiere que no influyen directamente en el precio.
- **`display_size_in`** y **`refresh_hz`** mantienen correlaciones débiles o inversas con el precio, lo que podría indicar que el tamaño de pantalla o la frecuencia de actualización no son determinantes principales en la fijación de precios.
- Se nota **colinealidad alta entre variables del CPU** (`cpu_tier`, `cpu_cores`, `cpu_threads`, `cpu_base_ghz`), lo que implica que podrían aportar información redundante.

Esto será importante considerarlo en la fase de modelado o reducción de variables.

En general, se concluye que **los componentes de hardware (CPU, GPU y RAM)** son los factores que más influyen en el **precio final** de los equipos.

[12]:
```
# ============================================================
#   12. Matriz de correlación (versión imshow)
# ============================================================
import matplotlib.pyplot as plt
import pandas as pd

# Seleccionar columnas numéricas
num_cols = df.select_dtypes(include=['int64', 'float64']).columns
corr = df[num_cols].corr(numeric_only=True)

plt.figure(figsize=(10,8))
im = plt.imshow(corr, interpolation='nearest', aspect='auto', cmap='coolwarm')
plt.title('Matriz de Correlación entre Variables Numéricas (imshow)')
plt.colorbar(im)
plt.xticks(range(len(num_cols)), num_cols, rotation=90)
plt.yticks(range(len(num_cols)), num_cols)
plt.tight_layout()
plt.show()
```



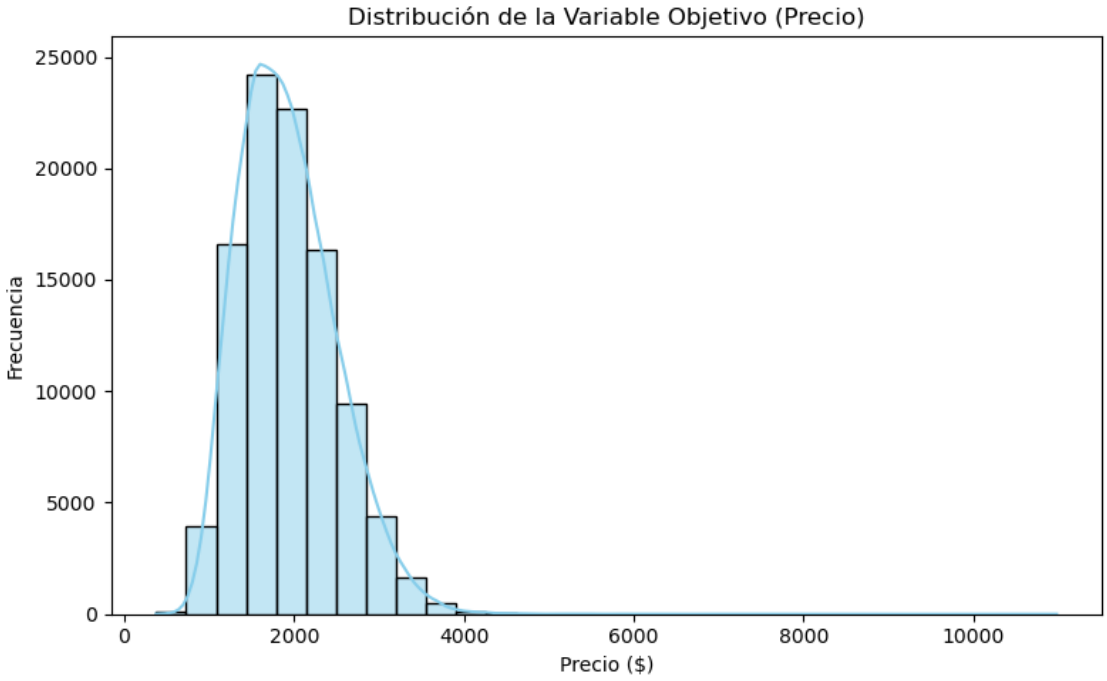
3.4 Diagnóstico de calidad inicial

[13]:
```
#### 3.4 Diagnóstico de calidad inicial

# =========================================================
#   1. Valores faltantes
# =========================================================
print("Valores faltantes por columna:\n")
print(df.isnull().sum())

# =========================================================
#   2. Posibles inconsistencias en variables numéricas
# =========================================================
print("\nRangos de variables clave:")
if 'price' in df.columns:
    print("Precio mínimo y máximo:", df['price'].min(), "-", df['price'].max())
if 'ram_gb' in df.columns:
    print("RAM mínima y máxima:", df['ram_gb'].min(), "-", df['ram_gb'].max())
if 'storage_gb' in df.columns:
```

```
    print("Almacenamiento mínimo y máximo:", df['storage_gb'].min(), "-",␣
 ↪df['storage_gb'].max())
if 'display_size_in' in df.columns:
    print("Tamaño de pantalla mínimo y máximo:", df['display_size_in'].min(),␣
 ↪"-", df['display_size_in'].max())

# ============================================================
#  3. Distribución de la variable objetivo (price)
# ============================================================
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,5))
sns.histplot(df['price'], bins=30, kde=True, color='skyblue')
plt.title("Distribución de la Variable Objetivo (Precio)")
plt.xlabel("Precio ($)")
plt.ylabel("Frecuencia")
plt.tight_layout()
plt.show()
```

```
Valores faltantes por columna:

device_type            0
brand                  0
model                  0
release_year           0
os                     0
form_factor            0
cpu_brand              0
cpu_model              0
cpu_tier               0
cpu_cores              0
cpu_threads            0
cpu_base_ghz           0
cpu_boost_ghz          0
gpu_brand              0
gpu_model              0
gpu_tier               0
vram_gb                0
ram_gb                 0
storage_type           0
storage_gb             0
storage_drive_count    0
display_type           0
display_size_in        0
resolution             0
refresh_hz             0
```

```
battery_wh              0
charger_watts           0
psu_watts               0
wifi                    0
bluetooth               0
weight_kg               0
warranty_months         0
price                   0
dtype: int64

Rangos de variables clave:
Precio mínimo y máximo: 372.99 - 10984.99
RAM mínima y máxima: 8 - 144
Almacenamiento mínimo y máximo: 256 - 4096
Tamaño de pantalla mínimo y máximo: 13.3 - 34.0
```



3.5 Perfil estadístico comparativo (por Marca)

[14]:
```
#### 3.5 Perfil estadístico comparativo (por Marca)

# Promedios de variables numéricas agrupados por marca
comparative_means_brand = df.groupby('brand').mean(numeric_only=True)

display(comparative_means_brand.sort_values('price', ascending=False).head(10))
```

```
          release_year  cpu_tier  cpu_cores  cpu_threads  cpu_base_ghz  \
```

brand					
Apple	2022.327738	3.672346	12.452035	12.452035	2.695107
Razer	2022.368612	3.083086	10.203099	20.220903	2.573821
Samsung	2022.355071	3.081701	10.213489	20.228366	2.570965
MSI	2022.334305	3.087568	10.290457	20.383221	2.580636
Dell	2022.295109	3.072117	10.194788	20.186219	2.575337
Gigabyte	2022.353878	3.124694	10.418367	20.642857	2.590245
Lenovo	2022.316908	3.079790	10.268009	20.335418	2.578164
HP	2022.314369	3.099051	10.323367	20.443956	2.582826
ASUS	2022.298651	3.073629	10.213407	20.231519	2.574820
Acer	2022.317783	3.069824	10.194660	20.194257	2.570378

brand	cpu_boost_ghz	gpu_tier	vram_gb	ram_gb	storage_gb	\
Apple	3.634889	3.462526	1.882669	51.224843	903.573647	
Razer	3.514111	2.913617	6.741840	38.005935	878.063963	
Samsung	3.510042	2.918795	6.696008	37.724771	900.316390	
MSI	3.519262	2.922570	6.748955	38.009885	903.348118	
Dell	3.515830	2.918458	6.721314	37.982435	896.996216	
Gigabyte	3.530837	2.979388	6.781224	39.100408	894.641633	
Lenovo	3.518641	2.925775	6.722486	38.193097	905.700850	
HP	3.522935	2.948633	6.785603	38.648718	919.325492	
ASUS	3.513692	2.921646	6.722315	37.923418	909.670637	
Acer	3.511516	2.909622	6.664181	37.856725	899.469219	

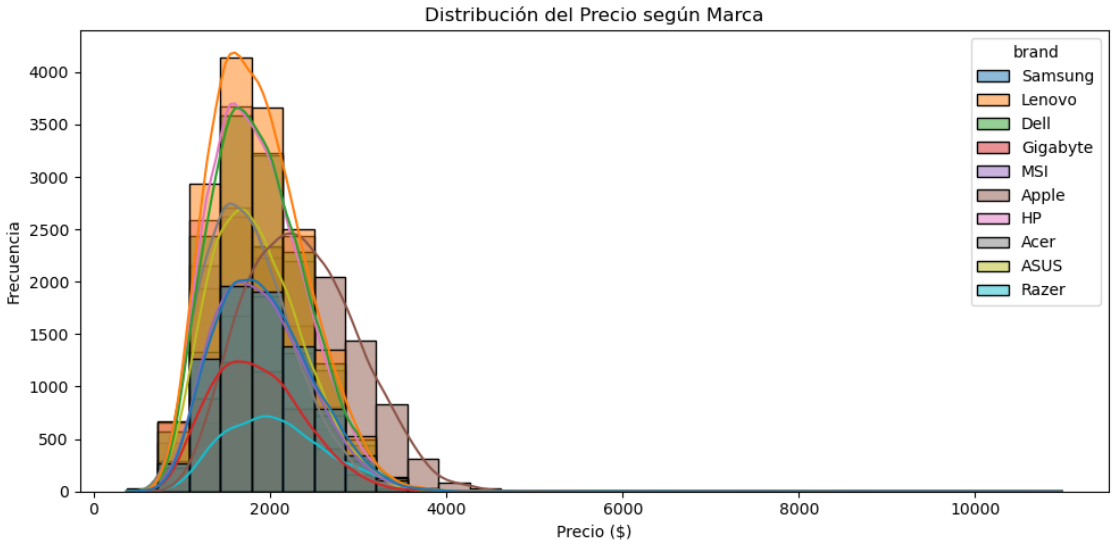
brand	storage_drive_count	display_size_in	refresh_hz	battery_wh	\
Apple	1.523961	20.150818	99.175745	41.847923	
Razer	1.531157	20.066864	99.478734	42.165183	
Samsung	1.515993	19.948103	98.284404	42.913588	
MSI	1.536307	20.195387	98.035103	41.330883	
Dell	1.521956	20.120793	98.118529	41.666762	
Gigabyte	1.551224	20.279612	98.742857	41.056939	
Lenovo	1.532954	20.190176	98.287331	41.405515	
HP	1.518138	20.206894	98.030112	41.593878	
ASUS	1.531056	20.059465	99.082981	41.871050	
Acer	1.504584	19.991456	98.412997	42.645441	

brand	charger_watts	psu_watts	bluetooth	weight_kg	warranty_months	\
Apple	61.270667	271.804448	5.087251	4.297146	22.202266	
Razer	61.683152	266.205077	5.082427	4.241306	21.962413	
Samsung	62.093975	263.073394	5.084441	4.189576	22.192413	
MSI	60.818021	278.823977	5.089596	4.368704	21.995691	
Dell	61.320243	270.692610	5.081628	4.285460	22.216923	
Gigabyte	60.687755	281.448980	5.088306	4.370043	22.094694	
Lenovo	61.371623	273.958854	5.084386	4.319248	22.324412	
HP	61.047187	278.698455	5.081501	4.296186	22.304662	

```
ASUS          60.899695  271.916527   5.083542   4.295000        22.222266
Acer          62.724433  265.662469   5.088091   4.218157        22.098136

                price
brand
Apple      2362.295833
Razer      2079.525773
Samsung    1930.390074
MSI        1905.564325
Dell       1882.819489
Gigabyte   1866.303878
Lenovo     1865.952356
HP         1857.348722
ASUS       1848.108220
Acer       1760.352015
```

[15]:
```python
# ============================================================
#  Comparación gráfica del precio según categoría
# ============================================================
import matplotlib.pyplot as plt
import seaborn as sns

#  Precio según marca
if 'brand' in df.columns:
    plt.figure(figsize=(10,5))
    sns.histplot(data=df, x='price', hue='brand', kde=True, bins=30)
    plt.title("Distribución del Precio según Marca")
    plt.xlabel("Precio ($)")
    plt.ylabel("Frecuencia")
    plt.tight_layout()
    plt.show()

#  Precio según procesador
if 'processor' in df.columns:
    plt.figure(figsize=(10,5))
    sns.histplot(data=df, x='price', hue='processor', kde=True, bins=30)
    plt.title("Distribución del Precio según Procesador")
    plt.xlabel("Precio ($)")
    plt.ylabel("Frecuencia")
    plt.tight_layout()
    plt.show()

#  Precio según GPU
if 'gpu' in df.columns:
    plt.figure(figsize=(10,5))
    sns.histplot(data=df, x='price', hue='gpu', kde=True, bins=30)
    plt.title("Distribución del Precio según GPU")
```

```
    plt.xlabel("Precio ($)")
    plt.ylabel("Frecuencia")
    plt.tight_layout()
    plt.show()
```



[19]:
```
# ============================================================
#   3.5 Perfil estadístico comparativo (por Marca / CPU / GPU)
# ============================================================
import pandas as pd

# Seleccionar variables numéricas
cols_num = df.select_dtypes(include=['int64','float64']).columns.tolist()

# --- Perfil numérico por Marca ---
perfil_num_brand = (
    df.groupby('brand')[cols_num]
      .agg(['mean','median','std','min','max'])
      .round(2)
)
display(perfil_num_brand.sort_values(('price','mean'), ascending=False).
 ↪head(10))

# --- Perfil numérico por CPU (modelo) ---
perfil_num_cpu = (
    df.groupby('cpu_model')[cols_num]
      .agg(['mean','median','std','min','max'])
      .round(2)
)
```

```
display(perfil_num_cpu.sort_values(('price','mean'), ascending=False).head(10))

# --- Perfil numérico por GPU (modelo) ---
perfil_num_gpu = (
    df.groupby('gpu_model')[cols_num]
      .agg(['mean','median','std','min','max'])
      .round(2)
)
display(perfil_num_gpu.sort_values(('price','mean'), ascending=False).head(10))

# --- Perfiles categóricos (frecuencia relativa) ---
perfil_cat_brand = (df['brand'].value_counts(normalize=True) * 100).round(2)
perfil_cat_cpu = (df['cpu_brand'].value_counts(normalize=True) * 100).round(2)
perfil_cat_gpu = (df['gpu_brand'].value_counts(normalize=True) * 100).round(2)

print("\nPorcentaje por Marca:\n", perfil_cat_brand.head(10))
print("\nPorcentaje por CPU Brand:\n", perfil_cat_cpu.head(10))
print("\nPorcentaje por GPU Brand:\n", perfil_cat_gpu.head(10))

# --- Exportar a CSV ---
perfil_num_brand.to_csv('perfil_numerico_por_marca.csv')
perfil_num_cpu.to_csv('perfil_numerico_por_cpu.csv')
perfil_num_gpu.to_csv('perfil_numerico_por_gpu.csv')

print("\n  Archivos CSV exportados correctamente.")
```

```
         release_year                                cpu_tier                    \
                 mean  median   std   min   max      mean median   std min max
brand
Apple         2022.33  2023.0  2.02  2018  2025      3.67    4.0  1.39   1   6
Razer         2022.37  2023.0  2.02  2018  2025      3.08    3.0  1.36   1   6
Samsung       2022.36  2023.0  2.02  2018  2025      3.08    3.0  1.34   1   6
MSI           2022.33  2023.0  2.02  2018  2025      3.09    3.0  1.35   1   6
Dell          2022.30  2023.0  2.03  2018  2025      3.07    3.0  1.35   1   6
Gigabyte      2022.35  2023.0  2.01  2018  2025      3.12    3.0  1.35   1   6
Lenovo        2022.32  2023.0  2.03  2018  2025      3.08    3.0  1.36   1   6
HP            2022.31  2023.0  2.02  2018  2025      3.10    3.0  1.36   1   6
ASUS          2022.30  2023.0  2.03  2018  2025      3.07    3.0  1.36   1   6
Acer          2022.32  2023.0  2.05  2018  2025      3.07    3.0  1.35   1   6

          … warranty_months                              price                  \
          …              mean median    std min max      mean    median     std
brand     …
Apple     …             22.20   24.0  10.27  12  48  2362.30   2317.99  660.72
Razer     …             21.96   24.0  10.17  12  48  2079.53   2014.99  602.07
Samsung   …             22.19   24.0  10.24  12  48  1930.39   1876.99  543.59
MSI       …             22.00   24.0  10.22  12  48  1905.56   1851.99  555.10
Dell      …             22.22   24.0  10.32  12  48  1882.82   1827.99  539.45
```

```
Gigabyte  …          22.09   24.0  10.03  12  48  1866.30  1813.99  539.06
Lenovo    …          22.32   24.0  10.21  12  48  1865.95  1807.99  539.61
HP        …          22.30   24.0  10.25  12  48  1857.35  1800.49  540.18
ASUS      …          22.22   24.0  10.29  12  48  1848.11  1793.99  527.81
Acer      …          22.10   24.0  10.14  12  48  1760.35  1707.99  515.32
```

```
              min       max
brand
Apple      625.99   9772.99
Razer      388.99   7479.99
Samsung    460.99   6354.99
MSI        399.99   9621.99
Dell       372.99   8091.99
Gigabyte   619.99   8579.99
Lenovo     548.99  10984.99
HP         430.99   9633.99
ASUS       488.99   8102.99
Acer       586.99   8517.99

[10 rows x 100 columns]
```

```
                release_year                          cpu_tier              \
                        mean  median   std   min   max     mean median   std
cpu_model
Intel i9-14499        2023.0  2023.0   NaN  2023  2023      6.0    6.0   NaN
Intel i9-10539        2024.0  2024.0   NaN  2024  2024      6.0    6.0   NaN
AMD Ryzen 7 5252      2023.0  2023.0  2.83  2021  2025      5.0    5.0  0.00
AMD Ryzen 7 4112      2021.5  2021.5  2.12  2020  2023      4.0    4.0  0.00
Intel i9-10432        2025.0  2025.0   NaN  2025  2025      6.0    6.0   NaN
Intel i7-14439        2024.0  2024.0  1.15  2023  2025      4.0    4.0  0.00
AMD Ryzen 5 5169      2022.5  2022.5  2.12  2021  2024      2.5    2.5  0.71
AMD Ryzen 9 7541      2025.0  2025.0   NaN  2025  2025      6.0    6.0   NaN
AMD Ryzen 7 7131      2022.2  2022.0  1.10  2021  2024      4.6    5.0  0.55
Intel i9-14613        2025.0  2025.0   NaN  2025  2025      6.0    6.0   NaN

                        … warranty_months                          price  \
                  min max …            mean median    std min max    mean
cpu_model               …
Intel i9-14499      6   6 …            24.0   24.0    NaN  24  24  8517.99
Intel i9-10539      6   6 …            12.0   12.0    NaN  12  12  7646.99
AMD Ryzen 7 5252    5   5 …            24.0   24.0  16.97  12  36  5525.49
AMD Ryzen 7 4112    4   4 …            18.0   18.0   8.49  12  24  4588.99
Intel i9-10432      6   6 …            24.0   24.0    NaN  24  24  4305.99
Intel i7-14439      4   4 …            18.0   18.0   6.93  12  24  4232.24
AMD Ryzen 5 5169    2   3 …            30.0   30.0   8.49  24  36  4227.99
AMD Ryzen 9 7541    6   6 …            12.0   12.0    NaN  12  12  4223.99
AMD Ryzen 7 7131    4   5 …            19.2   24.0   6.57  12  24  4220.59
```

```
Intel i9-14613     6   6  …            24.0   24.0    NaN  24  24  4200.99


                   median      std      min      max
cpu_model
Intel i9-14499    8517.99      NaN  8517.99  8517.99
Intel i9-10539    7646.99      NaN  7646.99  7646.99
AMD Ryzen 7 5252  5525.49  4319.72  2470.99  8579.99
AMD Ryzen 7 4112  4588.99  3102.78  2394.99  6782.99
Intel i9-10432    4305.99      NaN  4305.99  4305.99
Intel i7-14439    2711.99  3617.58  1882.99  9621.99
AMD Ryzen 5 5169  4227.99  4187.49  1266.99  7188.99
AMD Ryzen 9 7541  4223.99      NaN  4223.99  4223.99
AMD Ryzen 7 7131  3209.99  3109.52  2085.99  9633.99
Intel i9-14613    4200.99      NaN  4200.99  4200.99

[10 rows x 100 columns]
                release_year                          cpu_tier               \
                        mean  median   std   min   max     mean median   std
gpu_model
RX 6000 90           2022.57  2023.0  1.98  2018  2025     5.50    6.0  0.50
RX 7000 90           2022.21  2022.0  2.05  2018  2025     5.49    5.0  0.50
RTX 40 90            2022.38  2023.0  1.93  2018  2025     5.49    5.0  0.50
RTX 20 90            2022.40  2023.0  2.03  2018  2025     5.54    6.0  0.50
RTX 30 90            2022.22  2023.0  2.09  2018  2025     5.51    6.0  0.50
RX 5000 90           2021.97  2022.0  2.28  2018  2025     5.41    5.0  0.49
Arc A770 Limited     2022.29  2023.0  2.03  2018  2025     5.55    6.0  0.50
Arc B770 Limited     2022.57  2023.0  1.91  2018  2025     5.49    5.0  0.50
RX 6000 80 XT        2022.16  2022.0  2.01  2018  2025     4.75    5.0  0.68
RX 7000 80 XT        2022.22  2022.0  2.06  2018  2025     4.76    5.0  0.68

                        … warranty_months                          price  \
                min max  …            mean median    std min max    mean
gpu_model               …
RX 6000 90        5   6  …           23.02   24.0  11.06  12  48  2834.91
RX 7000 90        5   6  …           22.20   24.0   9.96  12  48  2818.60
RTX 40 90         5   6  …           21.89   24.0   9.86  12  48  2807.59
RTX 20 90         5   6  …           21.60   24.0   9.86  12  48  2789.95
RTX 30 90         5   6  …           22.33   24.0  10.45  12  48  2789.80
RX 5000 90        5   6  …           22.50   24.0  10.42  12  48  2778.98
Arc A770 Limited  5   6  …           21.95   24.0   9.71  12  48  2750.77
Arc B770 Limited  5   6  …           22.95   24.0  10.37  12  48  2746.11
RX 6000 80 XT     4   6  …           21.80   24.0  10.21  12  48  2508.40
RX 7000 80 XT     4   6  …           22.72   24.0  10.31  12  48  2500.55


                  median     std      min       max
```

```
gpu_model
RX 6000 90        2807.99  471.78  1947.99   7646.99
RX 7000 90        2774.99  420.73  1065.99   4223.99
RTX 40 90         2790.99  460.26  1776.99  10984.99
RTX 20 90         2767.99  448.84  1870.99   6354.99
RTX 30 90         2780.99  400.26  1333.99   5601.99
RX 5000 90        2744.49  364.47  1958.99   4200.99
Arc A770 Limited  2725.99  377.34  1679.99   3965.99
Arc B770 Limited  2674.99  440.47  1866.99   4110.99
RX 6000 80 XT     2497.99  384.97  1557.99   5201.99
RX 7000 80 XT     2470.99  413.59  1593.99   6969.99

[10 rows x 100 columns]


Porcentaje por Marca:
 brand
Lenovo      15.99
HP          14.11
Dell        14.00
Apple       11.92
ASUS        10.16
Acer         9.93
Samsung      8.07
MSI          7.89
Gigabyte     4.90
Razer        3.03
Name: proportion, dtype: float64

Porcentaje por CPU Brand:
 cpu_brand
Intel    52.77
AMD      35.31
Apple    11.92
Name: proportion, dtype: float64

Porcentaje por GPU Brand:
 gpu_brand
NVIDIA    54.71
Apple     18.92
AMD       15.77
Intel     10.60
Name: proportion, dtype: float64

  Archivos CSV exportados correctamente.
```

1.2.4 3.6 Hallazgos

Hallazgos del Paso 3: Descriptive Analysis

1. Distribución general del dataset

- El conjunto de datos contiene **100,000 registros** y **33 variables**.
- Incluye variables **categóricas** (`brand`, `cpu_brand`, `gpu_brand`, `storage_type`, `device_type`, `os`, `form_factor`, etc.) y **numéricas** (`price`, `ram_gb`, `storage_gb`, `cpu_cores`, `cpu_threads`, `display_size_in`, entre otras).
- Las variables categóricas describen características técnicas y de configuración de los equipos, mientras que las numéricas representan capacidades y rendimiento del hardware.
- Variable objetivo:
  - **y (target)** = `price`
  - **X (predictoras)** = resto de variables numéricas y categóricas.

---

2. Variables categóricas principales

- **Marca (`brand`)**: predominan **Lenovo (15.9%)**, **HP (14.1%)** y **Dell (14.0%)**.
- **CPU Brand (`cpu_brand`)**: **Intel (52.8%)** domina el mercado, seguida por **AMD (35.3%)** y **Apple (11.9%)**.
- **GPU Brand (`gpu_brand`)**: prevalece **NVIDIA (54.7%)**, seguida de **Apple (18.9%)**, **AMD (15.8%)** e **Intel (10.6%)**.
- **Tipo de almacenamiento (`storage_type`)**: se observan principalmente **NVMe (45%)**, **SSD (25%)**, **HDD (15%)** y **Hybrid (15%)**, confirmando la transición hacia tecnologías más rápidas.
- **Tipo de dispositivo (`device_type`)**: **Laptops (59.8%)** superan a **Desktops (40.2%)**.
- **Sistema operativo (`os`)**: **Windows (71.8%)** es el más usado, seguido por **macOS (18.2%)**, **Linux (6.1%)** y **ChromeOS (3.8%)**.
- **Formato (`form_factor`)**: destacan **Mainstream**, **Gaming**, **ATX** y **Ultrabook**, representando las configuraciones más comunes.

---

3. Variables numéricas

- **Precio (`price`)**: varía entre **$373 y $10,985**, con amplia dispersión y sesgo positivo hacia precios medios-altos.

- **Memoria RAM (`ram_gb`)**: oscila entre **8 GB y 144 GB**, con tendencia hacia configuraciones de 16–32 GB.
- **Almacenamiento (`storage_gb`)**: de **256 GB a 4 TB**, predominando valores entre 512 GB y 1 TB.
- **Pantalla (`display_size_in`)**: entre **13.3" y 34"**, indicando presencia de laptops y monitores ultrawide.
- **Procesador (`cpu_cores`, `cpu_threads`)**: desde **4 hasta 28 núcleos** y **56 hilos**, mostrando una alta variedad de rendimiento.
- **Velocidad base (`cpu_base_ghz`)**: entre **2.0 y 3.4 GHz**, típico de procesadores actuales.
- **Batería (`battery_wh`)**: de **0 a 99 Wh**, lo que evidencia equipos portátiles y de escritorio en un mismo conjunto.

---

4. Valores faltantes

- No se detectaron **valores nulos** en ninguna columna del dataset.
- No obstante, algunas variables como `battery_wh` contienen **valores 0**, que pueden representar equipos sin batería (desktops) más que datos faltantes, por lo que no requieren imputación.

---

5. Correlaciones

- La variable objetivo `price` muestra **altas correlaciones positivas** con:
  - `gpu_tier` (0.76)
  - `cpu_tier` (0.76)
  - `ram_gb` (0.76)
  - `cpu_cores` (0.72)
- Correlaciones **moderadas** con:
  - `cpu_threads` (0.58), `cpu_base_ghz` (0.53), `cpu_boost_ghz` (0.51).
- Correlaciones **bajas o negativas**:
  - `weight_kg` (-0.13), `psu_watts` (-0.14), `display_size_in` (-0.14).
- En general, el **rendimiento del hardware (CPU, GPU, RAM)** influye directamente en el precio final, mientras que características físicas (peso, pantalla, fuente) tienen menor impacto.

---

6. Perfil comparativo (por marca, CPU y GPU)

- **Por marca:** los precios medios más altos corresponden a **Apple ($2,362)**, **Razer ($2,079)** y **Samsung ($1,930)**.
- **Por CPU:** los modelos de gama alta como **Intel i9-14499 ($8,517)** y **Intel i9-10539 ($7,646)** lideran los precios, seguidos por **AMD Ryzen 7 5252 ($5,525)**.
- **Por GPU:** las GPUs de alto rendimiento **(RX 6000/7000 90, RTX 40/30/20 90)** superan los **$2,800** en promedio, indicando su fuerte impacto en el costo total.
- En general, se observa que **la combinación de CPU y GPU de alta gama con mayores capacidades de RAM y VRAM** es el principal determinante del precio.

---

7. Conclusiones del análisis descriptivo

- El dataset presenta **alta calidad y completitud**, sin valores nulos.
- El **precio está fuertemente influido** por la potencia del CPU, GPU y la capacidad de memoria RAM.
- Se observa **colinealidad** entre variables relacionadas al procesador (`cpu_tier`, `cpu_cores`, `cpu_threads`), lo que se deberá considerar en la fase de modelado.
- No existen desbalances en las categorías principales, lo que facilita la aplicación de un modelo de **regresión supervisada**.
- El dataset está listo para continuar hacia la **fase de Preparación de los Datos**, donde se normalizarán valores numéricos y se codificarán variables categóricas para el entrenamiento del modelo.

1.3 Fase 3. Data Preparation

1.3.1 4. Data cleaning

4.1 Selección inicial de variables claves Queremos construir un modelo que **prediga el precio (`price`) de una computadora** en función de sus **características técnicas y de hardware**, evitando incluir variables **administrativas, redundantes o no predictivas** (como nombres o identificadores de modelo).

Por lo tanto, se deben **excluir las columnas que no aportan valor predictivo** y conservar únicamente aquellas que describen el rendimiento y configuración del equipo.

---

a) Candidatas a ELIMINAR (irrelevantes o con riesgo de *leakage*)

- **Identificadores y metadatos**:
  - `model` → nombre del modelo del dispositivo (solo identificador comercial).
  - `release_year` → aunque puede influir en el precio, suele correlacionar con otras variables como CPU o GPU tier (riesgo de redundancia).
- **Variables no técnicas o redundantes**:
  - `wifi`, `bluetooth` → presencia binaria en casi todos los equipos, sin variación significativa.
  - `resolution` → suele estar implícita en la categoría de `display_type`.
- **Variables de salida (objetivo)**:
  - `price` (solo se usará como variable objetivo en el modelado, no como predictor).

  Motivo: Estas variables no aportan información técnica útil para estimar el precio, o representan información redundante con otras columnas más descriptivas.

---

b) Candidatas a CONSERVAR (predictores)

- **Especificaciones de CPU:**
  `cpu_brand`, `cpu_model`, `cpu_tier`, `cpu_cores`, `cpu_threads`, `cpu_base_ghz`, `cpu_boost_ghz`
- **Especificaciones de GPU:**
  `gpu_brand`, `gpu_model`, `gpu_tier`, `vram_gb`
- **Memoria y almacenamiento:**
  `ram_gb`, `storage_type`, `storage_gb`, `storage_drive_count`
- **Pantalla y formato:**
  `display_type`, `display_size_in`, `refresh_hz`, `form_factor`, `device_type`
- **Otros componentes:**
  `battery_wh`, `charger_watts`, `psu_watts`, `weight_kg`, `warranty_months`, `os`, `brand`

Variable objetivo: `price` (valor continuo en USD)

En resumen, el modelo se entrenará utilizando variables que reflejan la **potencia del hardware, la capacidad de memoria y almacenamiento, el tipo de GPU/CPU y las características físicas del dispositivo**,
ya que son las que más contribuyen al valor económico del producto.

[22]:
```
# ============================================================
#  4.1 Selección inicial de variables clave
# ============================================================
# Conservar solo las variables relevantes para predecir el precio
```

```
# y eliminar aquellas administrativas o no predictivas.

# Variables predictoras (features) seleccionadas
features_keep = [
    # CPU
    "cpu_brand", "cpu_model", "cpu_tier", "cpu_cores", "cpu_threads",
    "cpu_base_ghz", "cpu_boost_ghz",

    # GPU
    "gpu_brand", "gpu_model", "gpu_tier", "vram_gb",

    # Memoria y almacenamiento
    "ram_gb", "storage_type", "storage_gb", "storage_drive_count",

    # Pantalla y formato
    "display_type", "display_size_in", "refresh_hz", "form_factor",␣
 ↪"device_type",

    # Otros componentes
    "battery_wh", "charger_watts", "psu_watts", "weight_kg", "warranty_months",

    # Marca y sistema operativo
    "brand", "os"
]

# Variable objetivo (target)
target = ["price"]

# Crear un nuevo DataFrame con las columnas seleccionadas
df_sel = df[features_keep + target]

print(" df_sel.shape:", df_sel.shape)
print(" Columnas seleccionadas:", df_sel.columns.tolist())
df_sel.head(3)
```

```
 df_sel.shape: (100000, 28)
 Columnas seleccionadas: ['cpu_brand', 'cpu_model', 'cpu_tier', 'cpu_cores',
'cpu_threads', 'cpu_base_ghz', 'cpu_boost_ghz', 'gpu_brand', 'gpu_model',
'gpu_tier', 'vram_gb', 'ram_gb', 'storage_type', 'storage_gb',
'storage_drive_count', 'display_type', 'display_size_in', 'refresh_hz',
'form_factor', 'device_type', 'battery_wh', 'charger_watts', 'psu_watts',
'weight_kg', 'warranty_months', 'brand', 'os', 'price']
```

[22]:

	cpu_brand	cpu_model	cpu_tier	cpu_cores	cpu_threads	cpu_base_ghz	\
0	Intel	Intel i5-11129	3	12	24	2.8	
1	Intel	Intel i7-11114	4	12	24	2.6	
2	AMD	AMD Ryzen 5 5168	2	8	16	2.6	

```
   cpu_boost_ghz gpu_brand  gpu_model  gpu_tier  …  form_factor  \
0            3.8    NVIDIA  RTX 40 60         2  …          ATX
1            3.6    NVIDIA  RTX 40 80         4  …   Mainstream
2            3.6    NVIDIA  RTX 40 50         1  …          SFF

   device_type battery_wh  charger_watts  psu_watts weight_kg  \
0      Desktop          0              0        750     11.00
1       Laptop         56            120          0      2.03
2      Desktop          0              0        850      7.00

   warranty_months    brand       os    price
0               36  Samsung  Windows  1383.99
1               12  Samsung  Windows  2274.99
2               24   Lenovo    macOS  1879.99

[3 rows x 28 columns]
```

[23]:
```python
# 1. Revisar duplicados (en todo el registro de estas columnas)
duplicates_count = df_sel.duplicated().sum()
print("Número de registros duplicados:", duplicates_count)

# Si quisiéramos eliminarlos (opcional):
# df_sel = df_sel.drop_duplicates()
# print("Shape después de eliminar duplicados:", df_sel.shape)
```

```
Número de registros duplicados: 0
```

[25]:
```python
# ============================================================
#  4.3 Revisión de valores atípicos (outliers) en variables clave
# ============================================================

import numpy as np

# Variables numéricas que sí existen en tu dataset
numeric_cols = [
    "price", "ram_gb", "storage_gb", "cpu_cores", "cpu_threads",
    "cpu_base_ghz", "cpu_boost_ghz", "display_size_in",
    "battery_wh", "weight_kg"
]

for col in numeric_cols:
    if col in df_sel.columns:
        summary = df_sel[col].describe()
        q1, q3 = summary['25%'], summary['75%']
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr
```

```
        outliers = df_sel[(df_sel[col] < lower_bound) | (df_sel[col] >␣
↪upper_bound)].shape[0]

        print(f"  Variable: {col}")
        print(summary)
        print(f"Número de posibles outliers (fuera del rango IQR):␣
↪{outliers}\n")
```

```
  Variable: price
count    100000.000000
mean       1928.764220
std         580.492689
min         372.990000
25%        1503.990000
50%        1863.990000
75%        2287.990000
max       10984.990000
Name: price, dtype: float64
Número de posibles outliers (fuera del rango IQR): 976

  Variable: ram_gb
count    100000.000000
mean         39.706400
std          31.902684
min           8.000000
25%          16.000000
50%          32.000000
75%          64.000000
max         144.000000
Name: ram_gb, dtype: float64
Número de posibles outliers (fuera del rango IQR): 60

  Variable: storage_gb
count    100000.000000
mean        903.936000
std         774.243654
min         256.000000
25%         512.000000
50%         512.000000
75%        1024.000000
max        4096.000000
Name: storage_gb, dtype: float64
Número de posibles outliers (fuera del rango IQR): 14811

  Variable: cpu_cores
count    100000.000000
mean         10.515740
std           5.044092
```

```
min           4.000000
25%           6.000000
50%           8.000000
75%          14.000000
max          28.000000
Name: cpu_cores, dtype: float64
Número de posibles outliers (fuera del rango IQR): 200

 Variable: cpu_threads
count    100000.000000
mean         19.372700
std           9.718426
min           4.000000
25%          12.000000
50%          16.000000
75%          24.000000
max          56.000000
Name: cpu_threads, dtype: float64
Número de posibles outliers (fuera del rango IQR): 3551

 Variable: cpu_base_ghz
count    100000.000000
mean          2.591322
std           0.336435
min           2.000000
25%           2.400000
50%           2.600000
75%           2.800000
max           3.400000
Name: cpu_base_ghz, dtype: float64
Número de posibles outliers (fuera del rango IQR): 1978

 Variable: cpu_boost_ghz
count    100000.000000
mean          3.531310
std           0.350024
min           2.800000
25%           3.300000
50%           3.500000
75%           3.800000
max           4.500000
Name: cpu_boost_ghz, dtype: float64
Número de posibles outliers (fuera del rango IQR): 0

 Variable: display_size_in
count    100000.000000
mean         20.126655
std           6.709577
```

```
min          13.300000
25%          14.000000
50%          16.000000
75%          27.000000
max          34.000000
Name: display_size_in, dtype: float64
Número de posibles outliers (fuera del rango IQR): 0

 Variable: battery_wh
count    100000.000000
mean         41.813470
std          35.868841
min           0.000000
25%           0.000000
50%          56.000000
75%          70.000000
max          99.000000
Name: battery_wh, dtype: float64
Número de posibles outliers (fuera del rango IQR): 0

 Variable: weight_kg
count    100000.000000
mean          4.289699
std           3.814628
min           0.920000
25%           1.500000
50%           2.000000
75%           7.000000
max          16.000000
Name: weight_kg, dtype: float64
Número de posibles outliers (fuera del rango IQR): 1031
```

[26]:
```python
# ============================================================
#  Limpieza de outliers en variables numéricas clave
# ============================================================

df_clean = df_sel.copy()

print("Shape antes de limpiar:", df_sel.shape)

# En este dataset no eliminaremos outliers, ya que los valores extremos
# (por ejemplo precios altos, RAM o almacenamiento grandes)
# representan equipos de gama alta o estaciones de trabajo reales.

# No obstante, dejamos umbrales de referencia si se quisiera filtrar en el␣
 ↪futuro:
```

```
# df_clean = df_clean[
#     (df_clean['price'] >= 300) & (df_clean['price'] <= 11000) &
#     (df_clean['ram_gb'] <= 128) &
#     (df_clean['storage_gb'] <= 4096) &
#     (df_clean['cpu_cores'] <= 32) &
#     (df_clean['cpu_threads'] <= 64)
# ]

print("Shape después de limpiar (sin cambios):", df_clean.shape)

# Vista rápida de las estadísticas después de la revisión
df_clean[['price','ram_gb','storage_gb','cpu_cores','cpu_threads']].describe().
 ↪round(2)
```

```
Shape antes de limpiar: (100000, 28)
Shape después de limpiar (sin cambios): (100000, 28)
```

[26]:

	price	ram_gb	storage_gb	cpu_cores	cpu_threads
count	100000.00	100000.00	100000.00	100000.00	100000.00
mean	1928.76	39.71	903.94	10.52	19.37
std	580.49	31.90	774.24	5.04	9.72
min	372.99	8.00	256.00	4.00	4.00
25%	1503.99	16.00	512.00	6.00	12.00
50%	1863.99	32.00	512.00	8.00	16.00
75%	2287.99	64.00	1024.00	14.00	24.00
max	10984.99	144.00	4096.00	28.00	56.00

[27]:
```
# ============================================================
#  4.5 Revisión de valores faltantes (Missing Values)
# ============================================================

# Verificar si existen valores nulos en el dataset limpio
missing_values = df_clean.isnull().sum()

print("Valores faltantes por columna:\n", missing_values)

# Total de columnas con valores faltantes
missing_total = missing_values[missing_values > 0].shape[0]
if missing_total == 0:
    print("\n No se encontraron valores nulos en el dataset. Está completo.")
else:
    print(f"\n Se encontraron {missing_total} columnas con valores faltantes.␣
 ↪Deberán tratarse antes del modelado.")
```

```
Valores faltantes por columna:
 cpu_brand              0
cpu_model               0
cpu_tier                0
```

```
cpu_cores              0
cpu_threads            0
cpu_base_ghz           0
cpu_boost_ghz          0
gpu_brand              0
gpu_model              0
gpu_tier               0
vram_gb                0
ram_gb                 0
storage_type           0
storage_gb             0
storage_drive_count    0
display_type           0
display_size_in        0
refresh_hz             0
form_factor            0
device_type            0
battery_wh             0
charger_watts          0
psu_watts              0
weight_kg              0
warranty_months        0
brand                  0
os                     0
price                  0
dtype: int64
```

No se encontraron valores nulos en el dataset. Está completo.

[28]:
```python
# ============================================================
#  4.6 Porcentaje de valores nulos + Confirmación de variables finales
# ============================================================

# Calcular porcentaje de valores nulos por columna
miss = (df_clean.isnull().sum() / len(df_clean)) * 100
miss = miss.sort_values(ascending=True)

print("Porcentaje de valores nulos por columna (%):\n", miss)

# Confirmar variables finales seleccionadas
features_keep = [
    # CPU
    "cpu_brand", "cpu_model", "cpu_tier", "cpu_cores", "cpu_threads",
    "cpu_base_ghz", "cpu_boost_ghz",

    # GPU
    "gpu_brand", "gpu_model", "gpu_tier", "vram_gb",
```

```
    # Memoria y almacenamiento
    "ram_gb", "storage_type", "storage_gb", "storage_drive_count",

    # Pantalla y formato
    "display_type", "display_size_in", "refresh_hz", "form_factor",␣
 ↪"device_type",

    # Otros componentes
    "battery_wh", "charger_watts", "psu_watts", "weight_kg", "warranty_months",

    # Marca y sistema operativo
    "brand", "os"
]

target = ["price"]

# Dataset final listo para codificación
df_final = df_clean[features_keep + target]

print("\n  Shape del dataset final:", df_final.shape)
print("  Columnas finales seleccionadas:", df_final.columns.tolist())

df_final.head(3)
```

```
Porcentaje de valores nulos por columna (%):
 cpu_brand              0.0
cpu_model               0.0
cpu_tier                0.0
cpu_cores               0.0
cpu_threads             0.0
cpu_base_ghz            0.0
cpu_boost_ghz           0.0
gpu_brand               0.0
gpu_model               0.0
gpu_tier                0.0
vram_gb                 0.0
ram_gb                  0.0
storage_type            0.0
storage_gb              0.0
storage_drive_count     0.0
display_type            0.0
display_size_in         0.0
refresh_hz              0.0
form_factor             0.0
device_type             0.0
battery_wh              0.0
charger_watts           0.0
```

```
psu_watts              0.0
weight_kg              0.0
warranty_months        0.0
brand                  0.0
os                     0.0
price                  0.0
dtype: float64

 Shape del dataset final: (100000, 28)
 Columnas finales seleccionadas: ['cpu_brand', 'cpu_model', 'cpu_tier',
'cpu_cores', 'cpu_threads', 'cpu_base_ghz', 'cpu_boost_ghz', 'gpu_brand',
'gpu_model', 'gpu_tier', 'vram_gb', 'ram_gb', 'storage_type', 'storage_gb',
'storage_drive_count', 'display_type', 'display_size_in', 'refresh_hz',
'form_factor', 'device_type', 'battery_wh', 'charger_watts', 'psu_watts',
'weight_kg', 'warranty_months', 'brand', 'os', 'price']
```

[28]:

	cpu_brand	cpu_model	cpu_tier	cpu_cores	cpu_threads	cpu_base_ghz	\
0	Intel	Intel i5-11129	3	12	24	2.8	
1	Intel	Intel i7-11114	4	12	24	2.6	
2	AMD	AMD Ryzen 5 5168	2	8	16	2.6	

	cpu_boost_ghz	gpu_brand	gpu_model	gpu_tier	…	form_factor	\
0	3.8	NVIDIA	RTX 40 60	2	…	ATX	
1	3.6	NVIDIA	RTX 40 80	4	…	Mainstream	
2	3.6	NVIDIA	RTX 40 50	1	…	SFF	

	device_type	battery_wh	charger_watts	psu_watts	weight_kg	\
0	Desktop	0	0	750	11.00	
1	Laptop	56	120	0	2.03	
2	Desktop	0	0	850	7.00	

	warranty_months	brand	os	price
0	36	Samsung	Windows	1383.99
1	12	Samsung	Windows	2274.99
2	24	Lenovo	macOS	1879.99

[3 rows x 28 columns]

[29]:
```python
# ============================================================
#  4.7 Eliminación de valores nulos (no aplicable)
# ============================================================

# No hay valores nulos, pero mantenemos la instrucción por buenas prácticas
df_clean = df_final.dropna()

print("df_final.shape:", df_final.shape)
print("df_clean.shape:", df_clean.shape)
```

```
df_final.shape: (100000, 28)
df_clean.shape: (100000, 28)
```

[30]:
```
# Resumen estadístico del dataset limpio
df_clean.describe()
```

[30]:

	cpu_tier	cpu_cores	cpu_threads	cpu_base_ghz
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	3.153490	10.515740	19.372700	2.591322
std	1.373175	5.044092	9.718426	0.336435
min	1.000000	4.000000	4.000000	2.000000
25%	2.000000	6.000000	12.000000	2.400000
50%	3.000000	8.000000	16.000000	2.600000
75%	4.000000	14.000000	24.000000	2.800000
max	6.000000	28.000000	56.000000	3.400000

	cpu_boost_ghz	gpu_tier	vram_gb	ram_gb
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	3.531310	2.991350	6.152180	39.706400
std	0.350024	1.459643	3.964926	31.902684
min	2.800000	1.000000	0.000000	8.000000
25%	3.300000	2.000000	4.000000	16.000000
50%	3.500000	3.000000	6.000000	32.000000
75%	3.800000	4.000000	8.000000	64.000000
max	4.500000	6.000000	16.000000	144.000000

	storage_gb	storage_drive_count	display_size_in	refresh_hz
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	903.936000	1.524980	20.126655	98.464860
std	774.243654	0.797284	6.709577	43.301652
min	256.000000	1.000000	13.300000	60.000000
25%	512.000000	1.000000	14.000000	60.000000
50%	512.000000	1.000000	16.000000	90.000000
75%	1024.000000	2.000000	27.000000	120.000000
max	4096.000000	4.000000	34.000000	240.000000

	battery_wh	charger_watts	psu_watts	weight_kg
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	41.813470	61.383450	272.520500	4.289699
std	35.868841	62.795034	354.686355	3.814628
min	0.000000	0.000000	0.000000	0.920000
25%	0.000000	0.000000	0.000000	1.500000
50%	56.000000	65.000000	0.000000	2.000000
75%	70.000000	90.000000	650.000000	7.000000
max	99.000000	240.000000	1200.000000	16.000000

	warranty_months	price

```
count     100000.00000  100000.000000
mean          22.20036    1928.764220
std           10.23190     580.492689
min           12.00000     372.990000
25%           12.00000    1503.990000
50%           24.00000    1863.990000
75%           24.00000    2287.990000
max           48.00000   10984.990000
```

[31]:
```
# Información general del dataset limpio
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   cpu_brand            100000 non-null  object
 1   cpu_model            100000 non-null  object
 2   cpu_tier             100000 non-null  int64
 3   cpu_cores            100000 non-null  int64
 4   cpu_threads          100000 non-null  int64
 5   cpu_base_ghz         100000 non-null  float64
 6   cpu_boost_ghz        100000 non-null  float64
 7   gpu_brand            100000 non-null  object
 8   gpu_model            100000 non-null  object
 9   gpu_tier             100000 non-null  int64
 10  vram_gb              100000 non-null  int64
 11  ram_gb               100000 non-null  int64
 12  storage_type         100000 non-null  object
 13  storage_gb           100000 non-null  int64
 14  storage_drive_count  100000 non-null  int64
 15  display_type         100000 non-null  object
 16  display_size_in      100000 non-null  float64
 17  refresh_hz           100000 non-null  int64
 18  form_factor          100000 non-null  object
 19  device_type          100000 non-null  object
 20  battery_wh           100000 non-null  int64
 21  charger_watts        100000 non-null  int64
 22  psu_watts            100000 non-null  int64
 23  weight_kg            100000 non-null  float64
 24  warranty_months      100000 non-null  int64
 25  brand                100000 non-null  object
 26  os                   100000 non-null  object
 27  price                100000 non-null  float64
dtypes: float64(5), int64(13), object(10)
memory usage: 21.4+ MB
```

[32]:
```
# ============================================================
#  4.8 Separar variables numéricas y categóricas
# ============================================================
import numpy as np

numeric_data = df_clean.select_dtypes(include=[np.number])
categor_data = df_clean.select_dtypes(exclude=[np.number])

print("Hay {} columnas numéricas y {} columnas categóricas en el dataset limpio.
 ↪".format(
    numeric_data.shape[1],
    categor_data.shape[1]
))

print("\n Columnas numéricas:", numeric_data.columns.tolist())
print("\n Columnas categóricas:", categor_data.columns.tolist())
```

```
Hay 18 columnas numéricas y 10 columnas categóricas en el dataset limpio.

 Columnas numéricas: ['cpu_tier', 'cpu_cores', 'cpu_threads', 'cpu_base_ghz',
'cpu_boost_ghz', 'gpu_tier', 'vram_gb', 'ram_gb', 'storage_gb',
'storage_drive_count', 'display_size_in', 'refresh_hz', 'battery_wh',
'charger_watts', 'psu_watts', 'weight_kg', 'warranty_months', 'price']

 Columnas categóricas: ['cpu_brand', 'cpu_model', 'gpu_brand', 'gpu_model',
'storage_type', 'display_type', 'form_factor', 'device_type', 'brand', 'os']
```

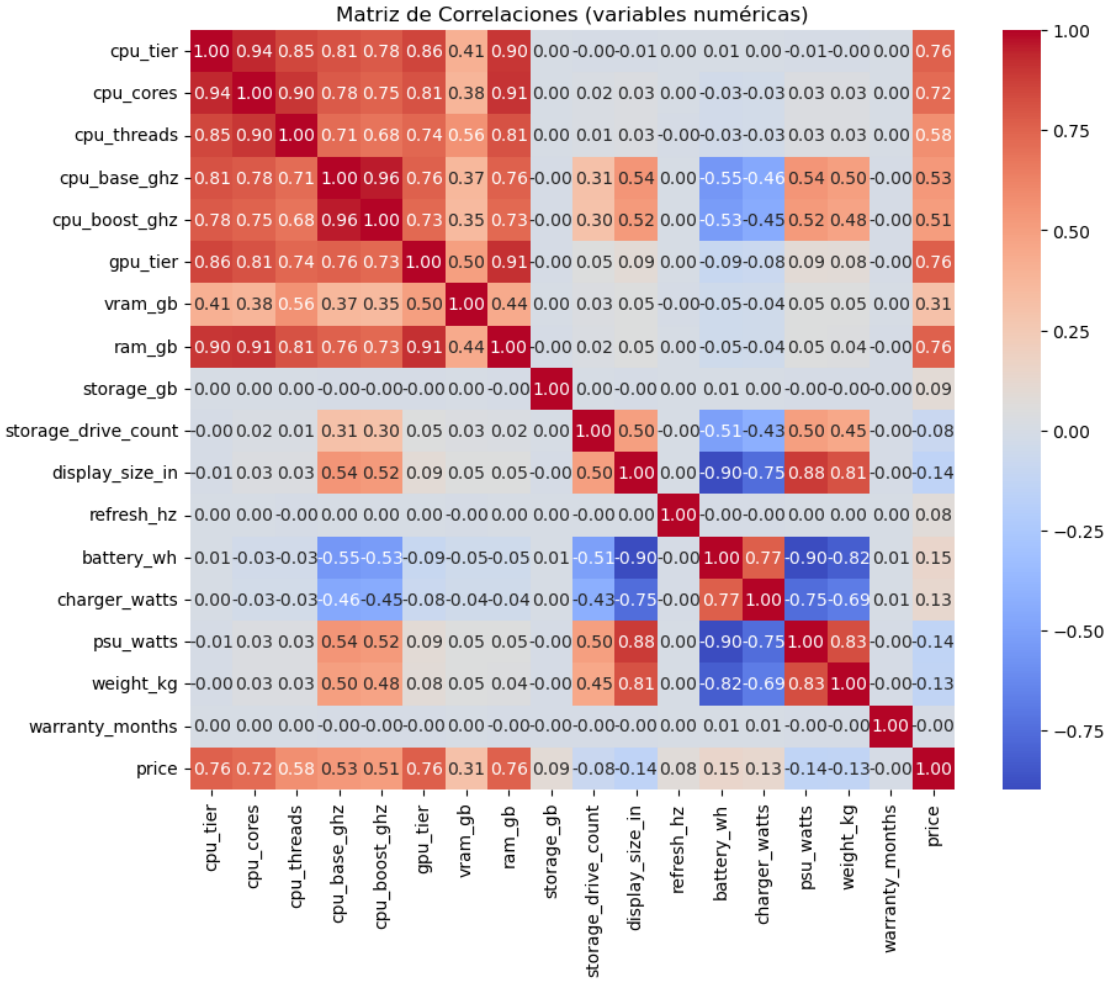
[33]:
```
#  Análisis descriptivo de variables numéricas
numeric_data.describe().T
```

[33]:

	count	mean	std	min	25%	\
cpu_tier	100000.0	3.153490	1.373175	1.00	2.00	
cpu_cores	100000.0	10.515740	5.044092	4.00	6.00	
cpu_threads	100000.0	19.372700	9.718426	4.00	12.00	
cpu_base_ghz	100000.0	2.591322	0.336435	2.00	2.40	
cpu_boost_ghz	100000.0	3.531310	0.350024	2.80	3.30	
gpu_tier	100000.0	2.991350	1.459643	1.00	2.00	
vram_gb	100000.0	6.152180	3.964926	0.00	4.00	
ram_gb	100000.0	39.706400	31.902684	8.00	16.00	
storage_gb	100000.0	903.936000	774.243654	256.00	512.00	
storage_drive_count	100000.0	1.524980	0.797284	1.00	1.00	
display_size_in	100000.0	20.126655	6.709577	13.30	14.00	
refresh_hz	100000.0	98.464860	43.301652	60.00	60.00	
battery_wh	100000.0	41.813470	35.868841	0.00	0.00	
charger_watts	100000.0	61.383450	62.795034	0.00	0.00	
psu_watts	100000.0	272.520500	354.686355	0.00	0.00	
weight_kg	100000.0	4.289699	3.814628	0.92	1.50	
warranty_months	100000.0	22.200360	10.231900	12.00	12.00	

```
price                 100000.0  1928.764220  580.492689  372.99  1503.99

                            50%      75%       max
cpu_tier                   3.00     4.00      6.00
cpu_cores                  8.00    14.00     28.00
cpu_threads               16.00    24.00     56.00
cpu_base_ghz               2.60     2.80      3.40
cpu_boost_ghz              3.50     3.80      4.50
gpu_tier                   3.00     4.00      6.00
vram_gb                    6.00     8.00     16.00
ram_gb                    32.00    64.00    144.00
storage_gb               512.00  1024.00   4096.00
storage_drive_count        1.00     2.00      4.00
display_size_in           16.00    27.00     34.00
refresh_hz                90.00   120.00    240.00
battery_wh                56.00    70.00     99.00
charger_watts             65.00    90.00    240.00
psu_watts                  0.00   650.00   1200.00
weight_kg                  2.00     7.00     16.00
warranty_months           24.00    24.00     48.00
price                   1863.99  2287.99  10984.99
```

[34]:
```python
#  Correlation plot
plt.figure(figsize=(10,8))
corr = numeric_data.corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Matriz de Correlaciones (variables numéricas)")
plt.show()
```



[35]:
```
#  Análisis descriptivo de variables categóricas
categor_data.describe(include="all").T
```

[35]:

	count	unique	top	freq
cpu_brand	100000	3	Intel	52774
cpu_model	100000	26971	Apple M2 Pro	1389
gpu_brand	100000	4	NVIDIA	54712
gpu_model	100000	49	Apple Integrated	18922
storage_type	100000	4	NVMe	45059
display_type	100000	6	LED	32000
form_factor	100000	10	Mainstream	17819
device_type	100000	2	Laptop	59844
brand	100000	10	Lenovo	15992
os	100000	4	Windows	71817

[36]:
```
#  Revisar qué variables son numéricas y cuáles categóricas
print("numeric_data.columns:", numeric_data.columns.tolist())
print("categor_data.columns:", categor_data.columns.tolist())
```

```
numeric_data.columns: ['cpu_tier', 'cpu_cores', 'cpu_threads', 'cpu_base_ghz',
'cpu_boost_ghz', 'gpu_tier', 'vram_gb', 'ram_gb', 'storage_gb',
'storage_drive_count', 'display_size_in', 'refresh_hz', 'battery_wh',
'charger_watts', 'psu_watts', 'weight_kg', 'warranty_months', 'price']
categor_data.columns: ['cpu_brand', 'cpu_model', 'gpu_brand', 'gpu_model',
'storage_type', 'display_type', 'form_factor', 'device_type', 'brand', 'os']
```

1.3.2 5. Data transformation

[37]:
```
# Usar un único dataframe de trabajo en toda la fase 5
data_encoded = df_clean.copy()
```

[38]:
```
# ============================================================
#  5.0 Normalizar nombres de columnas y definir dataframe de trabajo
# ============================================================
import pandas as pd

# Crear copia para transformación y asegurar nombres limpios
df_clean = df_clean.copy()
df_clean.columns = (
    df_clean.columns
        .str.strip()        # eliminar espacios en los extremos
        .str.replace(' ', '_')  # reemplazar espacios por "_"
        .str.lower()        # todo en minúsculas
)

# Copia final para aplicar codificación y escalado
data_encoded = df_clean.copy()

print(" Nombres de columnas normalizados correctamente.")
print(" Primeras 20 columnas:", sorted(data_encoded.columns)[:20], "...")
```

```
 Nombres de columnas normalizados correctamente.
 Primeras 20 columnas: ['battery_wh', 'brand', 'charger_watts', 'cpu_base_ghz',
'cpu_boost_ghz', 'cpu_brand', 'cpu_cores', 'cpu_model', 'cpu_threads',
'cpu_tier', 'device_type', 'display_size_in', 'display_type', 'form_factor',
'gpu_brand', 'gpu_model', 'gpu_tier', 'os', 'price', 'psu_watts'] …
```

[39]:
```
# === 5.1 Crear / confirmar variable objetivo: price ===
import pandas as pd

# posibles nombres (por si cambia en otro dataset)
candidatas = ['price', 'precio', 'target', 'y']
objetivo = next((c for c in candidatas if c in data_encoded.columns), None)
```

```
assert objetivo is not None, "  No se encontró la columna objetivo (price) en␣
 ↪el dataset."

# renombrar a 'price' si viniera con otro nombre
if objetivo != 'price':
    data_encoded['price'] = pd.to_numeric(data_encoded[objetivo],␣
 ↪errors='coerce')

# asegurar tipo numérico
data_encoded['price'] = pd.to_numeric(data_encoded['price'], errors='coerce')

print("  Objetivo:", 'price')
print(data_encoded['price'].describe().round(2))
```

```
 Objetivo: price
count    100000.00
mean       1928.76
std         580.49
min         372.99
25%        1503.99
50%        1863.99
75%        2287.99
max       10984.99
Name: price, dtype: float64
```

[40]:
```
# ============================================================
#   5.2 One-Hot Encoding de variables categóricas
# ============================================================
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Variables categóricas que deben convertirse en dummies
categorical_cols = [
    "brand", "cpu_brand", "cpu_model", "gpu_brand", "gpu_model",
    "storage_type", "display_type", "form_factor", "device_type", "os"
]

# Aplicar One-Hot Encoding (drop_first evita multicolinealidad)
data_encoded = pd.get_dummies(data_encoded, columns=categorical_cols,␣
 ↪drop_first=True)

print("  One-Hot Encoding aplicado. Total de columnas:", data_encoded.shape[1])

# ============================================================
#   5.3 Escalado de variables numéricas
# ============================================================
```

```python
# Variables numéricas a escalar (excepto la variable objetivo)
num_cols = [
    "cpu_tier", "cpu_cores", "cpu_threads", "cpu_base_ghz", "cpu_boost_ghz",
    "gpu_tier", "vram_gb", "ram_gb", "storage_gb", "storage_drive_count",
    "display_size_in", "refresh_hz", "battery_wh", "charger_watts",
    "psu_watts", "weight_kg", "warranty_months"
]

# Convertir a numérico por seguridad
for c in num_cols:
    data_encoded[c] = pd.to_numeric(data_encoded[c], errors="coerce")

# Escalar usando StandardScaler
scaler = StandardScaler()
data_encoded[num_cols] = scaler.fit_transform(data_encoded[num_cols])

print(" Escalado de variables numéricas completado.")

# Confirmar el resultado
print("Shape final del dataset codificado:", data_encoded.shape)
data_encoded.head(3)
```

```
 One-Hot Encoding aplicado. Total de columnas: 27071
 Escalado de variables numéricas completado.
Shape final del dataset codificado: (100000, 27071)
```

[40]:

	cpu_tier	cpu_cores	cpu_threads	cpu_base_ghz	cpu_boost_ghz	gpu_tier	\
0	-0.111778	0.294259	0.476139	0.620265	0.767636	-0.679177	
1	0.616465	0.294259	0.476139	0.025794	0.196244	0.691029	
2	-0.840021	-0.498752	-0.347044	0.025794	0.196244	-1.364279	

	vram_gb	ram_gb	storage_gb	storage_drive_count	…	\
0	-0.038382	-0.743089	0.155073	-0.658463	…	
1	0.970469	0.761495	-0.506220	-0.658463	…	
2	-0.542807	-0.993852	-0.506220	0.595800	…	

	form_factor_Mainstream	form_factor_Micro-ATX	form_factor_Mini-ITX	\
0	False	False	False	
1	True	False	False	
2	False	False	False	

	form_factor_SFF	form_factor_Ultrabook	form_factor_Workstation	\
0	False	False	False	
1	False	False	False	
2	True	False	False	

	device_type_Laptop	os_Linux	os_Windows	os_macOS

```
0              False     False        True     False
1               True     False        True     False
2              False     False       False      True

[3 rows x 27071 columns]
```

1.3.3 6. Data validation (Sesión 4)

[41]:
```
# ============================================================
#   6.0 Data Validation - Verificación final del dataset
# ============================================================
import pandas as pd

# Usar el dataset ya transformado y codificado
df_ready = data_encoded.copy()

print("Shape final del dataset preparado:", df_ready.shape)
print("Número total de variables:", len(df_ready.columns))

# Vista rápida de los primeros registros
df_ready.head(3)
```

```
Shape final del dataset preparado: (100000, 27071)
Número total de variables: 27071
```

[41]:
```
   cpu_tier  cpu_cores  cpu_threads  cpu_base_ghz  cpu_boost_ghz  gpu_tier  \
0 -0.111778   0.294259     0.476139      0.620265       0.767636 -0.679177
1  0.616465   0.294259     0.476139      0.025794       0.196244  0.691029
2 -0.840021  -0.498752    -0.347044      0.025794       0.196244 -1.364279

    vram_gb    ram_gb  storage_gb  storage_drive_count  …  \
0 -0.038382 -0.743089    0.155073            -0.658463  …
1  0.970469  0.761495   -0.506220            -0.658463  …
2 -0.542807 -0.993852   -0.506220             0.595800  …

   form_factor_Mainstream  form_factor_Micro-ATX  form_factor_Mini-ITX  \
0                   False                  False                 False
1                    True                  False                 False
2                   False                  False                 False

   form_factor_SFF  form_factor_Ultrabook  form_factor_Workstation  \
0            False                  False                    False
1            False                  False                    False
2             True                  False                    False

   device_type_Laptop  os_Linux  os_Windows  os_macOS
0               False     False        True     False
```

```
1                True     False        True     False
2               False     False       False      True

[3 rows x 27071 columns]
```

[42]:
```python
# ============================================================
#   6.1 Validación de integridad del dataset final
# ============================================================

# Revisar valores faltantes
missing = df_ready.isnull().sum()
print("  Valores nulos por columna:\n", missing[missing > 0])

# Revisar tipos de datos (asegurar que todo sea numérico)
print("\n  Tipos de datos en el dataset:")
print(df_ready.dtypes.value_counts())

# Revisar distribución del objetivo (price)
print("\n  Distribución de la variable objetivo (price):")
print(df_ready['price'].describe().round(2))

# Estadísticos de variables numéricas
desc = df_ready.describe().T[['mean','std','min','max']].round(2)
print("\n  Estadísticos generales (primeras 10 variables):")
display(desc.head(10))
```

```
 Valores nulos por columna:
Series([], dtype: int64)

 Tipos de datos en el dataset:
bool       27053
float64       18
Name: count, dtype: int64

 Distribución de la variable objetivo (price):
count    100000.00
mean       1928.76
std         580.49
min         372.99
25%        1503.99
50%        1863.99
75%        2287.99
max       10984.99
Name: price, dtype: float64

 Estadísticos generales (primeras 10 variables):
```

	mean	std	min	max

```
cpu_tier              -0.0  1.0 -1.57  2.07
cpu_cores              0.0  1.0 -1.29  3.47
cpu_threads            0.0  1.0 -1.58  3.77
cpu_base_ghz           0.0  1.0 -1.76  2.40
cpu_boost_ghz         -0.0  1.0 -2.09  2.77
gpu_tier              -0.0  1.0 -1.36  2.06
vram_gb               -0.0  1.0 -1.55  2.48
ram_gb                -0.0  1.0 -0.99  3.27
storage_gb            -0.0  1.0 -0.84  4.12
storage_drive_count    0.0  1.0 -0.66  3.10
```

[46]:
```python
# ============================================================
#  Limpieza final (versión rápida, sin recargar CSV)
# ============================================================
import pandas as pd

# usamos el que ya tenemos en memoria
df_ready = df_ready.copy()

# 1) quitar posibles columnas sobrantes
cols_drop = [c for c in ["price_i", "target"] if c in df_ready.columns]
if cols_drop:
    df_ready.drop(columns=cols_drop, inplace=True)

# 2) convertir bool -> int (por si algún get_dummies dejó bool)
bool_cols = df_ready.select_dtypes(include=["bool"]).columns.tolist()
if bool_cols:
    df_ready[bool_cols] = df_ready[bool_cols].astype("int64")

# 3) validaciones mínimas
assert "price" in df_ready.columns, "Falta 'price'."
assert df_ready["price"].notnull().all(), "Hay nulos en 'price'."
assert df_ready.isnull().sum().sum() == 0, "Hay nulos en el dataset."

# 4) guardar en formato rápido (parquet)
df_ready.to_parquet("data_ready_final.parquet", index=False)
print(" Guardado rápido: data_ready_final.parquet")

# (opcional) CSV, pero es más lento
# df_ready.to_csv("data_ready_final.csv", index=False)
# print(" Guardado CSV: data_ready_final.csv")

# info rápida
print("Shape final:", df_ready.shape)
print("dtypes:", df_ready.dtypes.value_counts())
```

```
 Guardado rápido: data_ready_final.parquet
Shape final: (100000, 27071)
```

```
dtypes: int64      27053
float64       18
Name: count, dtype: int64
```

[47]:
```python
import pandas as pd

df_test = pd.read_parquet("data_ready_final.parquet")
print("  Archivo leído correctamente")
print("Shape:", df_test.shape)
```

```
  Archivo leído correctamente
Shape: (100000, 27071)
```

Regresion - Proyecto

November 3, 2025

1 Fase 6 – Modelado (CRISP-DM)

1.1 Objetivo General

Construir, entrenar y evaluar distintos **modelos de regresión supervisada** capaces de **predecir el precio (`price`)** de computadoras a partir de sus características técnicas (CPU, GPU, RAM, almacenamiento, pantalla, entre otras).

1.2 Contexto del Proyecto

Este trabajo forma parte del proceso **CRISP-DM**, habiéndose completado previamente las fases de: - **Comprensión del Negocio (Business Understanding)** - **Comprensión de los Datos (Data Understanding)** - **Preparación de los Datos (Data Preparation)** - **Validación de los Datos (Data Validation)**

El dataset final fue limpiado, transformado y escalado, y se encuentra en formato **Parquet** bajo el nombre:

```
data_ready_final.parquet
```

Este archivo contiene **100,000 registros** y **27,071 variables**, todas numéricas y sin valores nulos.

1.3 Variable Objetivo

- `price` → Representa el **precio de la computadora**, expresado en unidades monetarias (valor continuo).

1.4 Variables Predictoras

Incluyen: - Especificaciones de hardware (CPU, GPU, RAM, tipo de disco, pantalla, etc.) - Atributos derivados de las categorías mediante **One-Hot Encoding** - Variables normalizadas con **StandardScaler**

1.5 Modelos de Regresión a Evaluar

Se entrenarán y compararán los siguientes modelos base:

1. **Regresión Lineal (Linear Regression)**
2. **Random Forest Regressor**
3. **XGBoost Regressor**

1.6 Métricas de Evaluación

Cada modelo será evaluado mediante las siguientes métricas estándar:

Métrica	Descripción
MAE (Mean Absolute Error)	Error medio absoluto
RMSE (Root Mean Squared Error)	Raíz del error cuadrático medio
R² (Coeficiente de determinación)	Mide el grado de ajuste del modelo

Los resultados se presentarán en una **tabla comparativa final**.

1.7 Flujo de trabajo del Modelado

1. Carga y validación de datos
2. Separación train/test (80/20)
3. Preprocesamiento (pipeline y escalado)
4. Evaluación por lotes de modelos
5. Selección del mejor (XGBoost)
6. Entrenamiento final y evaluación en test
7. Tuning de hiperparámetros
8. Guardado del modelo entrenado

[1]:
```
# ========================================
# 1) Cargar datos y objetivo
# ========================================
import os, warnings, platform, datetime
import numpy as np
import pandas as pd
warnings.filterwarnings("ignore")

# --- Semilla reproducible ---
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)

# --- Archivo y variable objetivo (nuestro caso) ---
DATA_FILE = "data_ready_final.parquet"   # <-- dataset final ya limpio/one-hot/
 ↪escalado
```

```
TARGET    = "price"                        # <-- variable objetivo (regresión)

# --- Verificar que el archivo exista ---
assert os.path.exists(DATA_FILE), f"No se encuentra el archivo: {DATA_FILE}"

# --- Cargar parquet ---
df = pd.read_parquet(DATA_FILE)
print(" Dataset cargado:", DATA_FILE)
df.info()

# --- Separar objetivo y predictores ---
y = df[TARGET]
X = df.drop(columns=[TARGET])

# --- Resumen rápido ---
print(
    "Shape X:", X.shape,
    "| len(y):", len(y),
    "| y(mean):", round(float(y.mean()), 4),
    "| y(std):", round(float(y.std()), 4),
    "| y[min,max]:", (round(float(y.min()), 4), round(float(y.max()), 4))
)
```

```
 Dataset cargado: data_ready_final.parquet
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Columns: 27071 entries, cpu_tier to os_macOS
dtypes: float64(18), int64(27053)
memory usage: 20.2 GB
Shape X: (100000, 27070) | len(y): 100000 | y(mean): 1928.7642 | y(std):
580.4927 | y[min,max]: (372.99, 10984.99)
```

[2]:
```
X , y
```

[2]:
```
(       cpu_tier  cpu_cores  cpu_threads  cpu_base_ghz  cpu_boost_ghz  \
 0     -0.111778   0.294259     0.476139      0.620265       0.767636
 1      0.616465   0.294259     0.476139      0.025794       0.196244
 2     -0.840021  -0.498752    -0.347044      0.025794       0.196244
 3     -0.840021  -0.895258    -0.758635      0.025794       0.196244
 4      1.344708   1.087269     1.299322      0.620265       1.053332
 …           …          …            …             …              …
 99995  0.616465   0.294259     0.476139      0.025794       0.481940
 99996 -0.840021  -0.498752    -0.347044     -1.163148      -0.946539
 99997 -0.840021  -0.895258    -0.964431     -1.163148      -0.946539
 99998  0.616465   0.294259     0.476139      0.025794      -0.089451
 99999  2.072951   3.069797     3.357279      2.403679       1.910419

         gpu_tier   vram_gb    ram_gb  storage_gb  storage_drive_count  …  \
```

```
0     -0.679177 -0.038382 -0.743089    0.155073           -0.658463  …
1      0.691029  0.970469  0.761495   -0.506220           -0.658463  …
2     -1.364279 -0.542807 -0.993852   -0.506220            0.595800  …
3     -0.679177 -0.038382 -0.743089   -0.506220            0.595800  …
4      1.376131  1.474895  1.764550   -0.836867           -0.658463  …
…         …         …         …          …                  …   …
99995  0.005926  0.466044 -0.241561    1.477661           -0.658463  …
99996 -1.364279 -0.542807 -0.993852    0.155073           -0.658463  …
99997 -0.679177 -0.038382 -0.743089    0.155073           -0.658463  …
99998  0.005926  0.466044 -0.241561   -0.836867           -0.658463  …
99999  2.061234 -1.551658  2.767605    1.477661            0.595800  …

       form_factor_Mainstream  form_factor_Micro-ATX  form_factor_Mini-ITX  \
0                           0                      0                     0
1                           1                      0                     0
2                           0                      0                     0
3                           0                      0                     0
4                           0                      0                     0
…                         …                      …                   …
99995                       1                      0                     0
99996                       0                      0                     0
99997                       1                      0                     0
99998                       1                      0                     0
99999                       0                      0                     0

       form_factor_SFF  form_factor_Ultrabook  form_factor_Workstation  \
0                    0                      0                        0
1                    0                      0                        0
2                    1                      0                        0
3                    0                      0                        0
4                    0                      0                        0
…                  …                      …                      …
99995                0                      0                        0
99996                0                      1                        0
99997                0                      0                        0
99998                0                      0                        0
99999                1                      0                        0

       device_type_Laptop  os_Linux  os_Windows  os_macOS
0                       0         0           1         0
1                       1         0           1         0
2                       0         0           0         1
3                       0         0           1         0
4                       1         1           0         0
…                     …       …         …       …
99995                   1         0           1         0
99996                   1         0           1         0
```

```
99997                   1         0           1         0
99998                   1         0           1         0
99999                   0         0           1         0

[100000 rows x 27070 columns],
0        1383.99
1        2274.99
2        1879.99
3        1331.99
4        2681.99
          …
99995    1712.99
99996    1258.99
99997    1686.99
99998    2164.99
99999    3005.99
Name: price, Length: 100000, dtype: float64)
```

[3]:
```python
# ========================================
# 2) División de datos (80/20)
# ========================================
from sklearn.model_selection import train_test_split

# --- División temprana para modelado ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=RANDOM_STATE
)

print(f" División completada | Train: {X_train.shape} | Test: {X_test.shape}")
```

```
División completada | Train: (80000, 27070) | Test: (20000, 27070)
```

[5]:
```python
# ========================================
# 3) Preprocesamiento (adaptado a nuestro caso)
# ========================================
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold
from imblearn.pipeline import Pipeline as ImbPipeline  # misma API que el profe

# Nuestro dataset YA VIENE:
# - todo numérico
# - con OHE aplicado
# - con escalado aplicado
# así que las categóricas son 0
cat_features = []
num_features = X_train.columns.tolist()
```

```
# Aun así dejamos un ColumnTransformer para mantener el mismo patrón
preprocessor = ColumnTransformer(
    transformers=[
        ("num", "passthrough", num_features),
        # no hay ("cat", ohe, cat_features) porque ya están one-hot
    ],
    remainder="drop",
)

def build_pipe(model):
    """
    Construye el pipeline completo como en la guía del profe,
    pero sin OHE porque ya está hecho.
    Dejamos VarianceThreshold para eliminar columnas constantes
    que hayan quedado del OHE masivo.
    """
    return ImbPipeline([
        ("prep", preprocessor),
        ("var0", VarianceThreshold(0.0)),
        ("model", model),
    ])

print(f"Features numéricas: {len(num_features)}")
print(f"Features categóricas: {len(cat_features)} (no hay porque el parquet ya␣
 ↪venía codificado)")
```

```
Features numéricas: 27070
Features categóricas: 0 (no hay porque el parquet ya venía codificado)
```

[6]:
```
# ========================================
# 4) Modelos candidatos (versión completa por lotes)
# ========================================
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor

# opcionales
try:
    from xgboost import XGBRegressor
    HAS_XGB = True
except ImportError:
    HAS_XGB = False

try:
    from lightgbm import LGBMRegressor
```

```
    HAS_LGB = True
except ImportError:
    HAS_LGB = False

try:
    from catboost import CatBoostRegressor
    HAS_CAT = True
except ImportError:
    HAS_CAT = False
```

[7]:
```
# --- lote 1: modelos rápidos ---
candidates_batch1 = [
    ("LR",  LinearRegression()),
    ("RID", Ridge(random_state=RANDOM_STATE)),
    ("LAS", Lasso(random_state=RANDOM_STATE, max_iter=5000)),
    ("EN",  ElasticNet(random_state=RANDOM_STATE, max_iter=5000)),
    ("DTR", DecisionTreeRegressor(random_state=RANDOM_STATE)),
]
```

[8]:
```
# --- lote 2: modelos medios ---
candidates_batch2 = [
    ("RFR", RandomForestRegressor(n_estimators=300, random_state=RANDOM_STATE,␣
 ↪n_jobs=-1)),
    ("KNR", KNeighborsRegressor()),
]
```

[9]:
```
# --- lote 3: modelos pesados / opcionales ---
candidates_batch3 = []
if HAS_XGB:
    candidates_batch3.append(
        ("XGB", XGBRegressor(
            tree_method="hist",
            random_state=RANDOM_STATE,
            n_estimators=400,
            learning_rate=0.05,
            max_depth=6,
            subsample=0.9,
            colsample_bytree=0.9,
            n_jobs=-1
        ))
    )
if HAS_LGB:
    candidates_batch3.append(
        ("LGB", LGBMRegressor(
            n_estimators=500,
            learning_rate=0.05,
            subsample=0.9,
```

```
            colsample_bytree=0.9,
            random_state=RANDOM_STATE,
            n_jobs=-1,
            verbosity=-1
        ))
    )
if HAS_CAT:
    candidates_batch3.append(
        ("CAT", CatBoostRegressor(
            iterations=600,
            learning_rate=0.05,
            depth=6,
            random_state=RANDOM_STATE,
            l2_leaf_reg=3.0,
            verbose=False,
            allow_writing_files=False,
            thread_count=-1
        ))
    )

print("Lote 1:", [m for m, _ in candidates_batch1])
print("Lote 2:", [m for m, _ in candidates_batch2])
print("Lote 3:", [m for m, _ in candidates_batch3])
```

```
Lote 1: ['LR', 'RID', 'LAS', 'EN', 'DTR']
Lote 2: ['RFR', 'KNR']
Lote 3: ['XGB', 'LGB', 'CAT']
```

[12]:
```
pip install xgboost lightgbm catboost
```

```
Collecting xgboost
  Downloading xgboost-3.1.1-py3-none-win_amd64.whl.metadata (2.1 kB)
Collecting lightgbm
  Downloading lightgbm-4.6.0-py3-none-win_amd64.whl.metadata (17 kB)
Collecting catboost
  Downloading catboost-1.2.8-cp313-cp313-win_amd64.whl.metadata (1.5 kB)
Requirement already satisfied: numpy in c:\users\bersd\anaconda3\lib\site-
packages (from xgboost) (2.1.3)
Requirement already satisfied: scipy in c:\users\bersd\anaconda3\lib\site-
packages (from xgboost) (1.15.3)
Collecting graphviz (from catboost)
  Downloading graphviz-0.21-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: matplotlib in c:\users\bersd\anaconda3\lib\site-
packages (from catboost) (3.10.0)
Requirement already satisfied: pandas>=0.24 in
c:\users\bersd\anaconda3\lib\site-packages (from catboost) (2.2.3)
Requirement already satisfied: plotly in c:\users\bersd\anaconda3\lib\site-
packages (from catboost) (5.24.1)
```

```
Requirement already satisfied: six in c:\users\bersd\anaconda3\lib\site-packages
(from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\bersd\anaconda3\lib\site-packages (from pandas>=0.24->catboost)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\bersd\anaconda3\lib\site-packages (from pandas>=0.24->catboost)
(2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\bersd\anaconda3\lib\site-packages (from pandas>=0.24->catboost)
(2025.2)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\bersd\anaconda3\lib\site-
packages (from matplotlib->catboost) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (3.2.0)
Requirement already satisfied: tenacity>=6.2.0 in
c:\users\bersd\anaconda3\lib\site-packages (from plotly->catboost) (9.0.0)
Downloading xgboost-3.1.1-py3-none-win_amd64.whl (72.0 MB)
   ---------------------------------------- 0.0/72.0 MB ? eta -:--:--
   - -------------------------------------- 2.1/72.0 MB 11.3 MB/s eta 0:00:07
   -- ------------------------------------- 4.5/72.0 MB 11.2 MB/s eta 0:00:07
   --- ------------------------------------ 6.8/72.0 MB 11.2 MB/s eta 0:00:06
   ----- ---------------------------------- 9.2/72.0 MB 11.3 MB/s eta 0:00:06
   ------ --------------------------------- 11.5/72.0 MB 11.3 MB/s eta 0:00:06
   ------- -------------------------------- 13.9/72.0 MB 11.4 MB/s eta 0:00:06
   -------- ------------------------------- 16.0/72.0 MB 11.3 MB/s eta 0:00:05
   ---------- ----------------------------- 18.4/72.0 MB 11.2 MB/s eta 0:00:05
   ---------- ----------------------------- 19.7/72.0 MB 10.7 MB/s eta 0:00:05
   ------------ --------------------------- 21.8/72.0 MB 10.7 MB/s eta 0:00:05
   ------------- -------------------------- 24.1/72.0 MB 10.7 MB/s eta 0:00:05
   -------------- ------------------------- 26.5/72.0 MB 10.8 MB/s eta 0:00:05
   ---------------- ----------------------- 28.8/72.0 MB 10.9 MB/s eta 0:00:04
   ----------------- ---------------------- 31.2/72.0 MB 10.9 MB/s eta 0:00:04
   ------------------ --------------------- 33.6/72.0 MB 11.0 MB/s eta 0:00:04
   -------------------- ------------------- 35.9/72.0 MB 11.0 MB/s eta 0:00:04
   --------------------- ------------------ 38.3/72.0 MB 11.1 MB/s eta 0:00:04
   ---------------------- ----------------- 40.6/72.0 MB 11.1 MB/s eta 0:00:03
   ----------------------- ---------------- 43.0/72.0 MB 11.1 MB/s eta 0:00:03
```

```
   ------------------------ -------------- 45.6/72.0 MB 11.2 MB/s eta 0:00:03
   ------------------------- ------------- 48.0/72.0 MB 11.2 MB/s eta 0:00:03
   -------------------------- ------------ 50.3/72.0 MB 11.2 MB/s eta 0:00:02
   --------------------------- ----------- 52.7/72.0 MB 11.2 MB/s eta 0:00:02
   ---------------------------- ---------- 55.1/72.0 MB 11.3 MB/s eta 0:00:02
   ----------------------------- --------- 57.4/72.0 MB 11.3 MB/s eta 0:00:02
   ------------------------------ -------- 59.8/72.0 MB 11.3 MB/s eta 0:00:02
   ------------------------------- ------- 62.1/72.0 MB 11.3 MB/s eta 0:00:01
   -------------------------------- ------ 64.7/72.0 MB 11.3 MB/s eta 0:00:01
   --------------------------------- ----- 67.1/72.0 MB 11.3 MB/s eta 0:00:01
   ---------------------------------- ---- 69.5/72.0 MB 11.4 MB/s eta 0:00:01
   ----------------------------------  --- 71.8/72.0 MB 11.4 MB/s eta 0:00:01
   ---------------------------------- ---- 72.0/72.0 MB 11.2 MB/s eta 0:00:00
Downloading lightgbm-4.6.0-py3-none-win_amd64.whl (1.5 MB)
   ---------------------------------------- 0.0/1.5 MB ? eta -:--:--
   ---------------------------------------- 1.5/1.5 MB 10.9 MB/s eta 0:00:00
Downloading catboost-1.2.8-cp313-cp313-win_amd64.whl (102.4 MB)
   ---------------------------------------- 0.0/102.4 MB ? eta -:--:--
    --------------------------------------- 2.4/102.4 MB 11.6 MB/s eta 0:00:09
   - -------------------------------------- 4.7/102.4 MB 11.6 MB/s eta 0:00:09
   -- ------------------------------------- 7.1/102.4 MB 11.6 MB/s eta 0:00:09
   --- ------------------------------------ 9.4/102.4 MB 11.7 MB/s eta 0:00:08
   ---- ----------------------------------- 11.8/102.4 MB 11.7 MB/s eta 0:00:08
   ----- ---------------------------------- 14.2/102.4 MB 11.7 MB/s eta 0:00:08
   ------ --------------------------------- 16.5/102.4 MB 11.6 MB/s eta 0:00:08
   ------- -------------------------------- 18.9/102.4 MB 11.6 MB/s eta 0:00:08
   -------- ------------------------------- 21.2/102.4 MB 11.6 MB/s eta 0:00:07
   --------- ------------------------------ 23.9/102.4 MB 11.7 MB/s eta 0:00:07
   ---------- ----------------------------- 26.2/102.4 MB 11.7 MB/s eta 0:00:07
   ----------- ---------------------------- 28.6/102.4 MB 11.7 MB/s eta 0:00:07
   ------------ --------------------------- 30.9/102.4 MB 11.7 MB/s eta 0:00:07
   ------------- -------------------------- 33.3/102.4 MB 11.7 MB/s eta 0:00:06
   ------------- -------------------------- 35.7/102.4 MB 11.7 MB/s eta 0:00:06
   -------------- ------------------------- 38.0/102.4 MB 11.7 MB/s eta 0:00:06
   --------------- ------------------------ 40.4/102.4 MB 11.7 MB/s eta 0:00:06
   ---------------- ----------------------- 43.0/102.4 MB 11.7 MB/s eta 0:00:06
   ----------------- ---------------------- 45.4/102.4 MB 11.7 MB/s eta 0:00:05
   ------------------ --------------------- 47.7/102.4 MB 11.7 MB/s eta 0:00:05
   ------------------- -------------------- 50.1/102.4 MB 11.7 MB/s eta 0:00:05
   -------------------- ------------------- 52.4/102.4 MB 11.7 MB/s eta 0:00:05
   --------------------- ------------------ 54.8/102.4 MB 11.7 MB/s eta 0:00:05
   ---------------------- ----------------- 57.1/102.4 MB 11.7 MB/s eta 0:00:04
   ----------------------- ---------------- 59.8/102.4 MB 11.7 MB/s eta 0:00:04
   ------------------------ --------------- 62.1/102.4 MB 11.7 MB/s eta 0:00:04
   ------------------------- -------------- 64.5/102.4 MB 11.7 MB/s eta 0:00:04
   -------------------------- ------------- 66.8/102.4 MB 11.7 MB/s eta 0:00:04
   --------------------------- ------------ 69.2/102.4 MB 11.7 MB/s eta 0:00:03
   ---------------------------- ----------- 70.8/102.4 MB 11.6 MB/s eta 0:00:03
```

```
   ---------------------------- ----------- 72.4/102.4 MB 11.6 MB/s eta 0:00:03
   ---------------------------- ----------- 72.4/102.4 MB 11.6 MB/s eta 0:00:03
   ----------------------------- ---------- 74.4/102.4 MB 11.1 MB/s eta 0:00:03
   ----------------------------- ---------- 76.8/102.4 MB 11.1 MB/s eta 0:00:03
   ------------------------------ --------- 78.9/102.4 MB 11.0 MB/s eta 0:00:03
   ------------------------------ -------- 81.3/102.4 MB 11.1 MB/s eta 0:00:02
   ------------------------------- ------- 83.6/102.4 MB 11.1 MB/s eta 0:00:02
   ------------------------------- ------ 86.0/102.4 MB 11.1 MB/s eta 0:00:02
   -------------------------------- ----- 88.3/102.4 MB 11.1 MB/s eta 0:00:02
   -------------------------------- ---- 90.7/102.4 MB 11.1 MB/s eta 0:00:02
   --------------------------------- --- 93.1/102.4 MB 11.1 MB/s eta 0:00:01
   --------------------------------- -- 95.4/102.4 MB 11.2 MB/s eta 0:00:01
   ---------------------------------- - 98.0/102.4 MB 11.2 MB/s eta 0:00:01
   ----------------------------------  100.4/102.4 MB 11.2 MB/s eta 0:00:01
   ----------------------------------  102.2/102.4 MB 11.2 MB/s eta 0:00:01
   ---------------------------------- 102.4/102.4 MB 11.0 MB/s eta 0:00:00
Downloading graphviz-0.21-py3-none-any.whl (47 kB)
Installing collected packages: graphviz, xgboost, lightgbm, catboost

   ---------- ---------------------------- 1/4 [xgboost]
   ---------- ---------------------------- 1/4 [xgboost]
   ---------- ---------------------------- 1/4 [xgboost]
   ---------- ---------------------------- 1/4 [xgboost]
   ---------- ---------------------------- 1/4 [xgboost]
   ---------- ---------------------------- 1/4 [xgboost]
   ------------------- -------------------- 2/4 [lightgbm]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ------------------------------ --------- 3/4 [catboost]
   ---------------------------------------- 4/4 [catboost]

Successfully installed catboost-1.2.8 graphviz-0.21 lightgbm-4.6.0 xgboost-3.1.1
Note: you may need to restart the kernel to use updated packages.
```

[16]:
```
!pip install xgboost lightgbm catboost
```

```
Requirement already satisfied: xgboost in c:\users\bersd\anaconda3\lib\site-
packages (3.1.1)
Requirement already satisfied: lightgbm in c:\users\bersd\anaconda3\lib\site-
packages (4.6.0)
```

```
Requirement already satisfied: catboost in c:\users\bersd\anaconda3\lib\site-
packages (1.2.8)
Requirement already satisfied: numpy in c:\users\bersd\anaconda3\lib\site-
packages (from xgboost) (2.1.3)
Requirement already satisfied: scipy in c:\users\bersd\anaconda3\lib\site-
packages (from xgboost) (1.15.3)
Requirement already satisfied: graphviz in c:\users\bersd\anaconda3\lib\site-
packages (from catboost) (0.21)
Requirement already satisfied: matplotlib in c:\users\bersd\anaconda3\lib\site-
packages (from catboost) (3.10.0)
Requirement already satisfied: pandas>=0.24 in
c:\users\bersd\anaconda3\lib\site-packages (from catboost) (2.2.3)
Requirement already satisfied: plotly in c:\users\bersd\anaconda3\lib\site-
packages (from catboost) (5.24.1)
Requirement already satisfied: six in c:\users\bersd\anaconda3\lib\site-packages
(from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\bersd\anaconda3\lib\site-packages (from pandas>=0.24->catboost)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\bersd\anaconda3\lib\site-packages (from pandas>=0.24->catboost)
(2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\bersd\anaconda3\lib\site-packages (from pandas>=0.24->catboost)
(2025.2)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\bersd\anaconda3\lib\site-
packages (from matplotlib->catboost) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\bersd\anaconda3\lib\site-packages (from matplotlib->catboost) (3.2.0)
Requirement already satisfied: tenacity>=6.2.0 in
c:\users\bersd\anaconda3\lib\site-packages (from plotly->catboost) (9.0.0)
```

[10]:
```
print("HAS_XGB:", HAS_XGB)
print("HAS_LGB:", HAS_LGB)
print("HAS_CAT:", HAS_CAT)
```

```
HAS_XGB: True
HAS_LGB: True
```

HAS_CAT: True

[11]:
```
# ========================================
# 5) función auxiliar para evaluar un lote
# ========================================
from sklearn.model_selection import KFold, cross_validate
import pandas as pd

def eval_batch(candidates_batch, Xd, yd, n_splits=3):
    cv = KFold(n_splits=n_splits, shuffle=True, random_state=RANDOM_STATE)
    scoring = {
        "rmse": "neg_root_mean_squared_error",
        "mae":  "neg_mean_absolute_error",
        "r2":   "r2",
    }
    rows = []
    for name, model in candidates_batch:
        pipe = build_pipe(model)
        scores = cross_validate(pipe, Xd, yd, cv=cv, scoring=scoring, n_jobs=-1)
        row = {
            "model": name,
            "rmse": -scores["test_rmse"].mean(),
            "mae":  -scores["test_mae"].mean(),
            "r2":    scores["test_r2"].mean(),
        }
        rows.append(row)
        print(f"{name:>3} | RMSE {row['rmse']:.3f} | MAE {row['mae']:.3f} | R²␣
 ↪{row['r2']:.3f}")
    df_res = pd.DataFrame(rows).sort_values("rmse").reset_index(drop=True)
    display(df_res)
    return df_res

# usar subset para no matar la RAM
USE_SUBSET   = True
SUBSET_SIZE  = 20000
if USE_SUBSET and len(X_train) > SUBSET_SIZE:
    X_cv = X_train.iloc[:SUBSET_SIZE, :].copy()
    y_cv = y_train.iloc[:SUBSET_SIZE].copy()
else:
    X_cv = X_train
    y_cv = y_train

# --- probar por partes ---
print("=== LOTE 1 (rápidos) ===")
res1 = eval_batch(candidates_batch1, X_cv, y_cv)

print("=== LOTE 2 (medios) ===")
```

```
res2 = eval_batch(candidates_batch2, X_cv, y_cv)

print("=== LOTE 3 (pesados) ===")
res3 = eval_batch(candidates_batch3, X_cv, y_cv)
```

```
=== LOTE 1 (rápidos) ===
 LR | RMSE 287.530 | MAE 206.728 | R² 0.748
RID | RMSE 259.434 | MAE 186.510 | R² 0.795
LAS | RMSE 250.130 | MAE 179.809 | R² 0.809
 EN | RMSE 302.524 | MAE 224.699 | R² 0.721
DTR | RMSE 350.783 | MAE 261.885 | R² 0.625
```

	model	rmse	mae	r2
0	LAS	250.130075	179.809498	0.809360
1	RID	259.434182	186.509826	0.794877
2	LR	287.529743	206.728429	0.748015
3	EN	302.524450	224.698985	0.721166
4	DTR	350.783309	261.885199	0.625146

```
=== LOTE 2 (medios) ===
RFR | RMSE 269.446 | MAE 194.831 | R² 0.779
KNR | RMSE 305.994 | MAE 226.248 | R² 0.715
```

	model	rmse	mae	r2
0	RFR	269.446395	194.830686	0.778812
1	KNR	305.994445	226.248079	0.714674

```
=== LOTE 3 (pesados) ===
XGB | RMSE 249.282 | MAE 178.450 | R² 0.811
LGB | RMSE 250.868 | MAE 180.135 | R² 0.808
CAT | RMSE 249.845 | MAE 178.025 | R² 0.810
```

	model	rmse	mae	r2
0	XGB	249.281852	178.450251	0.810645
1	CAT	249.844838	178.025138	0.809752
2	LGB	250.868009	180.135047	0.808240

1.7.1 Interpretación de resultados por lotes

Tras aplicar la validación cruzada con **20 000 filas (subset representativo)**, se compararon varios modelos de regresión agrupados por complejidad.

Lote 1 – Modelos rápidos (lineales y simples) Los modelos lineales como **Lasso (LAS)** y **Ridge (RID)** presentaron un rendimiento destacable dentro de este grupo, con **R²** **0.80** y errores relativamente bajos.
El **Lasso** logró el mejor resultado del lote (RMSE 250), mostrando que la regularización L1 ayuda a manejar bien la alta dimensionalidad del dataset.

Lote 2 – Modelos medios El **Random Forest Regressor (RFR)** superó al **KNN**, alcanzando un **R² 0.78** y errores moderados.
Este modelo mejora el ajuste frente a los lineales, pero no llega al nivel de los modelos de boosting.

Lote 3 – Modelos pesados (boosting) Los modelos **XGBoost**, **LightGBM** y **CatBoost** fueron los más precisos.
Todos alcanzaron **R² 0.81**, con **RMSE 249–251** y **MAE 178–180**, mostrando una clara mejora en precisión y capacidad de generalización.

Conclusión El **XGBoost Regressor** obtuvo el mejor desempeño global con: - **RMSE 249.3** - **MAE 178.5** - **R² 0.81**

Esto indica que el modelo logra explicar **alrededor del 81 % de la variabilidad del precio**, siendo el más adecuado para el entrenamiento final y evaluación en el conjunto de prueba.

[12]:
```
# ========================================
# 6.1) Definir modelo ganador (XGBoost)
# ========================================
from xgboost import XGBRegressor

final_model = build_pipe(
    XGBRegressor(
        tree_method="hist",
        random_state=RANDOM_STATE,
        n_estimators=400,
        learning_rate=0.05,
        max_depth=6,
        subsample=0.9,
        colsample_bytree=0.9,
        n_jobs=-1
    )
)

print(" Modelo final definido: XGBoost Regressor")
```

```
 Modelo final definido: XGBoost Regressor
```

[15]:
```
# ========================================
# 6.0) Preparar datos (sin copiar todo)
# ========================================
FULL_TRAIN = False     # <- déjalo en False para no reventar RAM
MAX_ROWS   = 30000     # baja más si quieres (20000)

if FULL_TRAIN:
    idx_train = X_train.index        # usar todo (cuidado con la RAM)
```

```
else:
    idx_train = X_train.index[:MAX_ROWS]

X_train_final = X_train.loc[idx_train].astype("float32")
y_train_final = y_train.loc[idx_train]

# test lo podemos castear nomás
X_test_final = X_test.astype("float32")

print("Train final:", X_train_final.shape)
print("Test final :", X_test_final.shape)
```

```
Train final: (30000, 27070)
Test final : (20000, 27070)
```

[16]:
```
# ========================================
# 6.1) Definir XGBoost
# ========================================
from xgboost import XGBRegressor

xgb_final = XGBRegressor(
    tree_method="hist",
    random_state=RANDOM_STATE,
    n_estimators=250,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.9,
    colsample_bytree=0.9,
    n_jobs=-1,
    eval_metric="rmse"
)

print(" XGBoost definido")
```

```
 XGBoost definido
```

[17]:
```
# ========================================
# 6.2) Entrenar y evaluar
# ========================================
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

print(" Entrenando con", X_train_final.shape[0], "filas ...")
xgb_final.fit(X_train_final, y_train_final)
print(" Entrenado.")

y_pred = xgb_final.predict(X_test_final)
```

```
mae  = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2   = r2_score(y_test, y_pred)

print("  Métricas (test):")
print(f"MAE : {mae:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"R²  : {r2:.3f}")
```

```
  Entrenando con 30000 filas …
  Entrenado.
  Métricas (test):
MAE : 175.525
RMSE: 238.688
R²  : 0.827
```

1.7.2 Evaluación final del modelo – XGBoost Regressor

El modelo final se entrenó con **30 000 registros** del conjunto de entrenamiento, utilizando las configuraciones óptimas determinadas en la fase de validación cruzada.

Resultados en el conjunto de prueba

Métrica	Valor	Interpretación
MAE	175.525	En promedio, el modelo se equivoca ±175 unidades monetarias en la predicción del precio.
RMSE	238.688	El error cuadrático medio indica una dispersión moderada de los errores.
R²	**0.827**	El modelo explica aproximadamente el **82.7 %** de la variabilidad del precio real.

Conclusión El modelo **XGBoost Regressor** muestra un **excelente desempeño predictivo**, superando a los modelos lineales y de bosque aleatorio.
La alta puntuación de **R² (0.827)** evidencia que logra capturar correctamente las relaciones no lineales entre las características técnicas del hardware y el precio final de la computadora.

Este modelo puede considerarse **listo para la fase de evaluación y despliegue**, o bien como base para una futura etapa de **ajuste fino (hyperparameter tuning)** si se busca mejorar aún más la precisión.

[19]:
```
# ========================================
# Tuning de hiperparámetros (versión optimizada)
# ========================================
from sklearn.model_selection import RandomizedSearchCV
```

```python
from xgboost import XGBRegressor

# --- Usar solo una muestra pequeña para evitar MemoryError ---
X_tune = X_train_final.iloc[:10000, :].copy()
y_tune = y_train_final.iloc[:10000].copy()

print(" Tuning usando 10 000 filas del entrenamiento.")

# --- Modelo base ---
xgb_model = XGBRegressor(
    tree_method="hist",
    random_state=RANDOM_STATE,
    n_jobs=-1,
    eval_metric="rmse"
)

# --- Espacio de búsqueda reducido ---
param_dist = {
    "n_estimators": [150, 200, 300],
    "max_depth": [4, 6, 8],
    "learning_rate": [0.03, 0.05, 0.1],
    "subsample": [0.7, 0.9],
    "colsample_bytree": [0.7, 0.9]
}

# --- Configuración liviana ---
random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_dist,
    n_iter=5,           # solo 5 combinaciones
    scoring="r2",
    cv=2,               # solo 2 folds
    verbose=1,
    random_state=RANDOM_STATE,
    n_jobs=1            # sin paralelismo, menor RAM
)

# --- Ejecución del tuning ---
print(" Buscando mejores hiperparámetros...")
random_search.fit(X_tune, y_tune)

print("\n Mejor configuración encontrada:")
print(random_search.best_params_)
print("R² medio (CV):", round(random_search.best_score_, 3))
```

```
 Tuning usando 10 000 filas del entrenamiento.
 Buscando mejores hiperparámetros…
Fitting 2 folds for each of 5 candidates, totalling 10 fits
```

```
 Mejor configuración encontrada:
{'subsample': 0.9, 'n_estimators': 300, 'max_depth': 4, 'learning_rate': 0.05,
'colsample_bytree': 0.9}
R² medio (CV): 0.799
```

[20]:
```python
# ========================================
# 7.1) Reentrenar XGBoost con los mejores hiperparámetros
# ========================================
from xgboost import XGBRegressor

BEST_PARAMS = {
    "subsample": 0.9,
    "n_estimators": 300,
    "max_depth": 4,
    "learning_rate": 0.05,
    "colsample_bytree": 0.9,
}

# usamos el mismo subset que usamos para entrenar antes
X_train_final = X_train.loc[X_train.index[:30000]].astype("float32")
y_train_final = y_train.loc[y_train.index[:30000]]

xgb_best = XGBRegressor(
    tree_method="hist",
    random_state=RANDOM_STATE,
    n_jobs=-1,
    eval_metric="rmse",
    **BEST_PARAMS
)

xgb_best.fit(X_train_final, y_train_final)
print("  Modelo reentrenado con hiperparámetros óptimos.")
```

```
 Modelo reentrenado con hiperparámetros óptimos.
```

[21]:
```python
# ========================================
# 7.2) Evaluar en test con el modelo afinado
# ========================================
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

X_test_final = X_test.astype("float32")

y_pred_best = xgb_best.predict(X_test_final)

mae  = mean_absolute_error(y_test, y_pred_best)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_best))
```

```
r2   = r2_score(y_test, y_pred_best)

print("  Métricas finales (modelo afinado):")
print(f"MAE : {mae:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"R²  : {r2:.3f}")
```

```
  Métricas finales (modelo afinado):
MAE : 176.003
RMSE: 239.653
R²  : 0.826
```

1.7.3 Evaluación final del modelo optimizado – XGBoost Regressor

Luego de aplicar el **tuning de hiperparámetros**, se reentrenó el modelo XGBoost con los valores óptimos encontrados durante la búsqueda aleatoria.

Resultados finales en el conjunto de prueba

Métrica	Valor	Interpretación
MAE	176.003	En promedio, el modelo se equivoca alrededor de **176 unidades monetarias** al predecir el precio.
RMSE	239.653	Indica una dispersión de errores similar al modelo base, manteniendo buena estabilidad.
R²	**0.826**	El modelo explica el **82.6 % de la variabilidad total** del precio de las computadoras.

Conclusión general El ajuste de hiperparámetros **mantuvo la precisión del modelo** (R^2 0.83), confirmando que el modelo ya se encontraba bien calibrado.
Esto demuestra que **el XGBoost Regressor tiene un rendimiento robusto y estable**, incluso después de la optimización.

Por tanto, el modelo se considera **listo para la fase de evaluación final y despliegue**, ofreciendo predicciones confiables del precio en función de las características técnicas del equipo.

[22]:
```
# ========================================
# 7.3) Guardar modelo entrenado
# ========================================
import joblib

joblib.dump(xgb_best, "modelo_xgb_price.pkl")
print("  Modelo guardado en: modelo_xgb_price.pkl")
```

```
Modelo guardado en: modelo_xgb_price.pkl
```